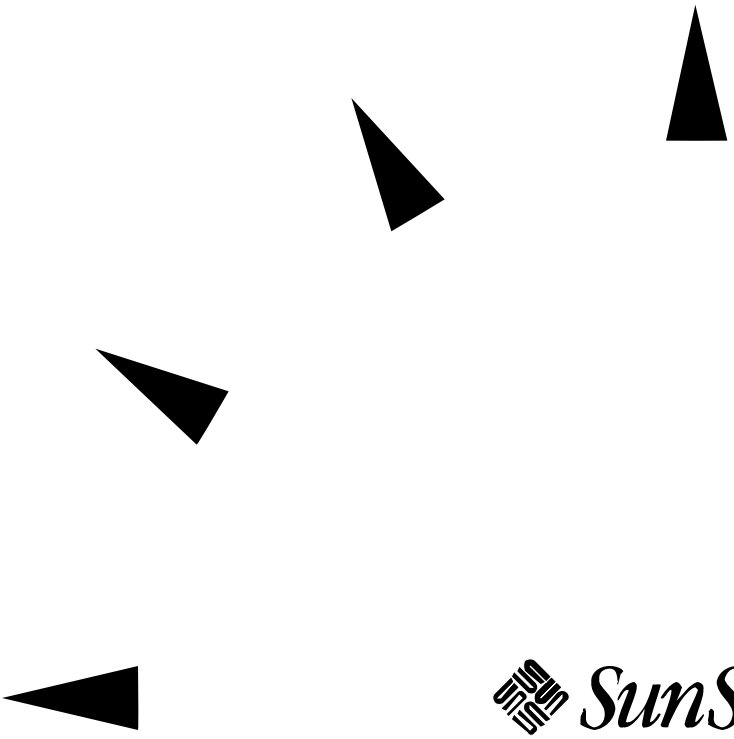
SunOS Reference Manual



Sun Microsystems, Inc.
2550 Garcia Avenue
Mountain View, CA 94043
U.S.A.

SunSoft
A Sun Microsystems, Inc. Business

2550 Garcia Avenue, Mountain View, California 94043-1100 U.S.A.

Preface

OVERVIEW

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

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

- Section 1 describes, in alphabetical order, commands available with the operating system.

- Section 1M describes, in alphabetical order, commands that are used chiefly for system maintenance and administration purposes.

- Section 2 describes all of the system calls. Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible returned value.

- Section 3 describes functions found in various libraries, other than those functions that directly invoke UNIX system primitives, which are described in Section 2 of this volume.

- Section 4 outlines the formats of various files. The C structure declarations for the file formats are given where applicable.

- Section 5 contains miscellaneous documentation such as character set tables, etc.

- Section 7 describes various special files that refer to specific hardware peripherals, and device drivers. STREAMS software drivers, modules and the STREAMS-generic set of system calls are also described.

- Section 9 provides reference information needed to write device drivers in the kernel operating systems environment. It describes two device driver interface specifications: the Device Driver Interface (DDI) and the Driver–Kernel Interface (DKI).

- Section 9E describes the DDI/DKI, DDI-only, and DKI-only entry-point routines a developer may include in a device driver.

- Section 9F describes the kernel functions available for use by device drivers.

- Section 9S describes the data structures used by drivers to share information between the driver and the kernel.

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

NAME

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

SYNOPSIS

This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full pathname is shown. Literal characters (commands and options) are in **bold** font and variables (arguments, parameters and substitution characters) are in *italic* font. Options and arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required.

The following special characters are used in this section:

[ ] The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument *must* be specified.

... Ellipses. Several values may be provided for the previous argument, or the previous argument can be specified multiple times, for example, '*filename* ...'.

| Separator. Only one of the arguments separated by this character can be specified at time.

PROTOCOL

This section occurs only in subsection 3R to indicate the protocol description file. The protocol specification pathname is always listed in **bold** font.

AVAILABILITY

This section briefly states any limitations on the availabilty of the command. These limitations could be hardware or software specific.

A specification of a class of hardware platform, such as **x86** or **SPARC**, denotes that the command or interface is applicable for the hardware platform specified.

In Section 1 and Section 1M, **AVAILABILITY** indicates which package contains the command being described on the manual page. In order to use the command, the specified package must have been installed with the operating system. If the package was not installed, see **pkgadd**(1) for information on how to upgrade.

MT-LEVEL

This section lists the **MT-LEVEL** of the library functions described in the Section 3 manual pages. The **MT-LEVEL** defines the libraries' ability to support threads. See **Intro**(3) for more information.

DESCRIPTION

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

IOCTLS

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

OPTIONS

This 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.

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 as **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 is provided as a *guidance* on use. This section lists special rules, features and commands that require in-depth explanations. The subsections listed below 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 super-user,

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

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

FILES

This section lists all filenames 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.

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. Messages appear in **bold** font with the exception of variables, which are in *italic* font.

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.

NAME

Intro, intro – introduction to file formats

DESCRIPTION

This section outlines the formats of various files. The C structure declarations for the file formats are given where applicable. Usually, the headers containing these structure declarations can be found in the directories **/usr/include** or **/usr/include/sys**. For inclusion in C language programs, however, the syntax **#include** <*filename.h*> or **#include** <**sys/***filename.h*> should be used.

Because the operating system now allows the existence of multiple file system types, there are several instances of multiple manual pages with the same name. These pages all display the name of the FSType to which they pertain in the form name_ *fstype* at the top of the page. For example, **fs_ufs**(4)

Name	Appears on Page	Description
/proc	**proc**(4)	process file system
.environ	**environ**(4)	user-preference variables files for AT&T FACE
.ott	**ott**(4)	FACE object architecture information
.pref	**environ**(4)	user-preference variables files for AT&T FACE
.rhosts	**hosts.equiv**(4)	trusted remote hosts and users
.variables	**environ**(4)	user-preference variables files for AT&T FACE
a.out	**a.out**(4)	Executable and Linking Format (ELF) files
acct	**acct**(4)	per-process accounting file format
addresses	**aliases**(4)	addresses and aliases for sendmail
admin	**admin**(4)	installation defaults file
aliases	**aliases**(4)	addresses and aliases for sendmail
ar	**ar**(4)	archive file format
archives	**archives**(4)	device header
asetenv	**asetenv**(4)	ASET environment file
asetmasters	**asetmasters**(4)	ASET master files
audit.log	**audit.log**(4)	audit trail file
audit_class	**audit_class**(4)	audit class definitions
audit_control	**audit_control**(4)	control information for system audit daemon
audit_data	**audit_data**(4)	current information on audit daemon
audit_event	**audit_event**(4)	audit event definition and class mapping
audit_user	**audit_user**(4)	per-user auditing data file
bootparams	**bootparams**(4)	boot parameter data base
cdtoc	**cdtoc**(4)	CD-ROM table of contents file
cklist.high	**asetmasters**(4)	ASET master files
cklist.low	**asetmasters**(4)	ASET master files
cklist.med	**asetmasters**(4)	ASET master files
clustertoc	**clustertoc**(4)	cluster table of contents description file

compver	**compver**(4)	compatible versions file
copyright	**copyright**(4)	copyright information file
core	**core**(4)	core image file
default_fs	**default_fs**(4)	specify the default file system type for local or remote file systems
depend	**depend**(4)	software dependencies file
device.cfinfo	**device.cfinfo**(4)	devconfig configuration files
device_allocate	**device_allocate**(4)	device_allocate file
device_maps	**device_maps**(4)	device_maps file
dfstab	**dfstab**(4)	file containing commands for sharing resources across a network
dir	**dir_ufs**(4)	format of ufs directories
dir_ufs	**dir_ufs**(4)	format of ufs directories
dirent	**dirent**(4)	file system independent directory entry
driver.conf	**driver.conf**(4)	driver configuration files
dumpdates	**ufsdump**(4)	incremental dump format
eisa	**sysbus**(4)	configuration files for ISA, EISA, and MCA bus device drivers
environ	**environ**(4)	user-preference variables files for AT&T FACE
ethers	**ethers**(4)	Ethernet address to hostname database or domain
fbtab	**logindevperm**(4)	login-based device permissions
fd	**fd**(4)	file descriptor files
filehdr	**filehdr**(4)	file header for common object files
format.dat	**format.dat**(4)	disk drive configuration for the format command.
forward	**aliases**(4)	addresses and aliases for sendmail
fs	**default_fs**(4)	specify the default file system type for local or remote file systems
fs_ufs	**fs_ufs**(4)	format of a ufs file system volume
fspec	**fspec**(4)	format specification in text files
fstypes	**fstypes**(4)	file that registers distributed file system packages
group	**group**(4)	group file
holidays	**holidays**(4)	prime / nonprime table for the accounting system
hosts.equiv	**hosts.equiv**(4)	trusted remote hosts and users
hosts	**hosts**(4)	host name database
inetd.conf	**inetd.conf**(4)	Internet servers database
init.d	**init.d**(4)	initialization and termination scripts for changing init states
inittab	**inittab**(4)	script for init
inode	**fs_ufs**(4)	format of a ufs file system volume
inode_ufs	**fs_ufs**(4)	format of a ufs file system volume

isa	**sysbus**(4)	configuration files for ISA, EISA, and MCA bus device drivers
issue	**issue**(4)	issue identification file
keytables	**keytables**(4)	keyboard table descriptions for loadkeys and dumpkeys
krb.conf	**krb.conf**(4)	Kerberos configuration file
krb.realms	**krb.realms**(4)	host to Kerberos realm translation file
limits	**limits**(4)	header for implementation-specific constants
loadfont	**loadfont**(4)	format of a font file used as input to the loadfont utility
logindevperm	**logindevperm**(4)	login-based device permissions
loginlog	**loginlog**(4)	log of failed login attempts
magic	**magic**(4)	file command's magic number file
mca	**sysbus**(4)	configuration files for ISA, EISA, and MCA bus device drivers
mnttab	**mnttab**(4)	mounted file system table
netconfig	**netconfig**(4)	network configuration database
netgroup	**netgroup**(4)	list of network groups
netid	**netid**(4)	netname database
netmasks	**netmasks**(4)	network mask database
netrc	**netrc**(4)	file for ftp remote login data
networks	**networks**(4)	network name database
nisfiles	**nisfiles**(4)	NIS+ database files and directory structure
nsswitch.conf	**nsswitch.conf**(4)	configuration file for the name-service switch
order	**order**(4)	package installation order description file
ott	**ott**(4)	FACE object architecture information
packagetoc	**packagetoc**(4)	package table of contents description file
passwd	**passwd**(4)	password file
path_to_inst	**path_to_inst**(4)	device instance number file
pathalias	**pathalias**(4)	alias file for FACE
phones	**phones**(4)	remote host phone number database
pkginfo	**pkginfo**(4)	package characteristics file
pkgmap	**pkgmap**(4)	package contents description file
plot	**plot**(4B)	graphics interface
pref	**environ**(4)	user-preference variables files for AT&T FACE
proc	**proc**(4)	process file system
profile	**profile**(4)	setting up an environment for user at login time
protocols	**protocols**(4)	protocol name database
prototype	**prototype**(4)	package information file
pseudo	**pseudo**(4)	configuration files for pseudo device drivers
publickey	**publickey**(4)	public key database

queuedefs	**queuedefs**(4)	queue description file for at, batch, and cron
remote	**remote**(4)	remote host description file
resolv.conf	**resolv.conf**(4)	configuration file for name server routines
rhosts	**hosts.equiv**(4)	trusted remote hosts and users
rmmount.conf	**rmmount.conf**(4)	removable media mounter configuration file
rmtab	**rmtab**(4)	remote mounted file system table
routing	**routing**(4)	system support for packet network routing
rpc	**rpc**(4)	rpc program number data base
rpld.conf	**rpld.conf**(4)	Remote Program Load (RPL) server configuration file
rt_dptbl	**rt_dptbl**(4)	real-time dispatcher parameter table
sbus	**sbus**(4)	configuration files for SBus device drivers
sccsfile	**sccsfile**(4)	format of an SCCS history file
scsi	**scsi**(4)	configuration files for SCSI target drivers
services	**services**(4)	Internet services and aliases
shadow	**shadow**(4)	shadow password file
sharetab	**sharetab**(4)	shared file system table
space	**space**(4)	disk space requirement file
strftime	**strftime**(4)	language specific strings
sysbus	**sysbus**(4)	configuration files for ISA, EISA, and MCA bus device drivers
syslog.conf	**syslog.conf**(4)	configuration file for syslogd system log daemon
system	**system**(4)	system configuration information file
term	**term**(4)	format of compiled term file
terminfo	**terminfo**(4)	terminal capability database
TIMEZONE	**TIMEZONE**(4)	set default system time zone and locale
timezone	**timezone**(4)	default timezone data base
ts_dptbl	**ts_dptbl**(4)	time-sharing dispatcher parameter table
ttydefs	**ttydefs**(4)	file contains terminal line settings information for ttymon
ttysrch	**ttysrch**(4)	directory search list for ttyname
tune.high	**asetmasters**(4)	ASET master files
tune.low	**asetmasters**(4)	ASET master files
tune.med	**asetmasters**(4)	ASET master files
ufsdump	**ufsdump**(4)	incremental dump format
uid_aliases	**asetmasters**(4)	ASET master files
unistd	**unistd**(4)	header for symbolic constants
updaters	**updaters**(4)	configuration file for NIS updating
utmp	**utmp**(4)	utmp and wtmp entry formats
utmpx	**utmpx**(4)	utmpx and wtmpx entry formats
variables	**environ**(4)	user-preference variables files for AT&T FACE
vfstab	**vfstab**(4)	table of file system defaults

vme	**vme**(4)	configuration files for VMEbus device drivers
vold.conf	**vold.conf**(4)	Volume Management configuration file
wtmp	**utmp**(4)	utmp and wtmp entry formats
wtmpx	**utmpx**(4)	utmpx and wtmpx entry formats
ypfiles	**ypfiles**(4)	Network Information Service Version 2, formerly knows as YP

NAME

TIMEZONE – set default system time zone and locale

SYNOPSIS

/etc/TIMEZONE
/etc/default/init

DESCRIPTION

This file sets the time zone environment variable **TZ**, and the locale-related environment variables **LANG**, **LC_COLLATE**, **LC_CTYPE**, **LC_MESSAGES**, **LC_MONETARY**, **LC_NUMERIC**, and **LC_TIME**.

/etc/TIMEZONE is a symbolic link to **/etc/default/init**.

The number of environments that can be set from **/etc/default/init** is limited to 20.

SEE ALSO

init(1M), **ctime**(3C), **environ**(5)

NAME

a.out – Executable and Linking Format (ELF) files

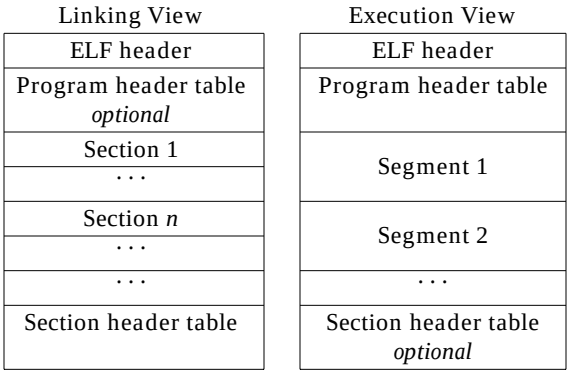
SYNOPSIS

#include <elf.h>

DESCRIPTION

The file name **a.out** is the default output file name from the link editor, **ld**(1). The link editor will make an **a.out** executable if there were no errors in linking. The output file of the assembler, **as**(1), also follows the format of the **a.out** file although its default file name is different.

Programs that manipulate ELF files may use the library that **elf**(3E) describes. An overview of the file format follows. For more complete information, see the references given below.



An ELF header resides at the beginning and holds a ‘‘road map’’ describing the file’s organization. Sections hold the bulk of object file information for the linking view: instructions, data, symbol table, relocation information, and so on. Segments hold the object file information for the program execution view. As shown, a segment may contain one or more sections.

A program header table, if present, tells the system how to create a process image. Files used to build a process image (execute a program) must have a program header table; relocatable files do not need one. A section header table contains information describing the file’s sections. Every section has an entry in the table; each entry gives information such as the section name, the section size, etc. Files used during linking must have a section header table; other object files may or may not have one.

Although the figure shows the program header table immediately after the ELF header, and the section header table following the sections, actual files may differ. Moreover, sections and segments have no specified order. Only the ELF header has a fixed position in the file.

When an **a.out** file is loaded into memory for execution, three logical segments are set up: the text segment, the data segment (initialized data followed by uninitialized, the latter actually being initialized to all 0’s), and a stack. The text segment is not writable by the program; if other processes are executing the same **a.out** file, the processes will share a single text segment.

The data segment starts at the next maximal page boundary past the last text address. If the system supports more than one page size, the ''maximal page'' is the largest supported size. When the process image is created, the part of the file holding the end of text and the beginning of data may appear twice. The duplicated chunk of text that appears at the beginning of data is never executed; it is duplicated so that the operating system may bring in pieces of the file in multiples of the actual page size without having to realign the beginning of the data section to a page boundary. Therefore, the first data address is the sum of the next maximal page boundary past the end of text plus the remainder of the last text address divided by the maximal page size. If the last text address is a multiple of the maximal page size, no duplication is necessary. The stack is automatically extended as required. The data segment is extended as requested by the **brk**(2) system call.

SEE ALSO

as(1), **cc**(1B), **ld**(1), **brk**(2), **elf**(3E)

ANSI C Programmer's Guide

NAME

acct – per-process accounting file format

SYNOPSIS

```
#include <sys/types.h>
#include <sys/acct.h>
```

DESCRIPTION

Files produced as a result of calling **acct**(2) have records in the form defined by **<sys/acct.h>**, whose contents are:

```
typedef ushort  comp_t;         /* pseudo "floating point" representation */
                                /* 3 bit base-8 exponent in the high */
                                /* order bits, and a 13-bit fraction */
                                /* in the low order bits. */

struct  acct
{
        char    ac_flag;        /* Accounting flag */
        char    ac_stat;        /* Exit status */
        uid_t   ac_uid;         /* Accounting user ID */
        gid_t   ac_gid;         /* Accounting group ID */
        dev_t   ac_tty;         /* control tty */
        time_t  ac_btime;       /* Beginning time */
        comp_t  ac_utime;       /* accounting user time in clock */
                                /* ticks */
        comp_t  ac_stime;       /* accounting system time in clock */
                                /* ticks */
        comp_t  ac_etime;       /* accounting total elapsed time in clock */
                                /* ticks */
        comp_t  ac_mem;         /* memory usage in clicks (pages) */
        comp_t  ac_io;          /* chars transferred by read/write */
        comp_t  ac_rw;          /* number of block reads/writes */
        char    ac_comm[8];     /* command name */
};

/*
 * Accounting Flags
 */
#define AFORK   01              /* has executed fork, but no exec */
#define ASU     02              /* used super-user privileges */
#define ACCTF   0300            /* record type */
#define AEXPND 040              /* Expanded Record Type – default */
```

In **ac_flag**, the **AFORK** flag is turned on by each **fork** and turned off by an **exec**. The **ac_comm** field is inherited from the parent process and is reset by any **exec**. Each time the system charges the process with a clock tick, it also adds to **ac_mem** the current process size, computed as follows:

(data size) + (text size) / (number of in-core processes using text)

The value of **ac_mem / (ac_stime + ac_utime)** can be viewed as an approximation to the mean process size, as modified by text sharing.

The structure **tacct**, (which resides with the source files of the accounting commands), represents a summary of accounting statistics for the user id **ta_uid**. This structure is used by the accounting commands to report statistics based on user id.

```
/*
 * total accounting (for acct period), also for day
 */
struct  tacct {
        uid_t           ta_uid;         /* user id */
        char            ta_name[8];     /* login name */
        float           ta_cpu[2];      /* cum. cpu time in minutes, */
                                        /* p/np (prime/non-prime time) */
        float           ta_kcore[2];    /* cum. kcore-minutes, p/np */
        float           ta_con[2];      /* cum. connect time in minutes, */
                                        /* p/np */
        float           ta_du;          /* cum. disk usage (blocks)*/
        long            ta_pc;          /* count of processes */
        unsigned short  ta_sc;          /* count of login sessions */
        unsigned short  ta_dc;          /* count of disk samples */
        unsigned short  ta_fee;         /* fee for special services */
};
```

ta_cpu, **ta_kcore**, and **ta_con** contain usage information pertaining to prime time and non-prime time hours. The first element in each array represents the time the resource was used during prime time hours. The second element in each array represents the time the resource was used during non-prime time hours. Prime time and non-prime time hours may be set in the **holidays** file (see **holidays**(4)).

ta_kcore is a cumulative measure of the amount of memory used over the accounting period by processes owned by the user with uid **ta_uid**. The amount shown represents kilobyte segments of memory used, per minute.

ta_con represents the amount of time the user was logged in to the system.

FILES

/etc/acct/holidays prime / non-prime time table

SEE ALSO

acctcom(1), **acct**(1M), **acctcon**(1M), **acctmerg**(1M), **acctprc**(1M), **acctsh**(1M), **prtacct**(1M), **runacct**(1M), **shutacct**(1M), **acct**(2), **exec**(2), **fork**(2)

NOTES

The **ac_mem** value for a short-lived command gives little information about the actual size of the command, because **ac_mem** may be incremented while a different command (for example, the shell) is being executed by the process.

NAME

admin – installation defaults file

DESCRIPTION

admin is a generic name for an ASCII file that defines default installation actions by assigning values to installation parameters. For example, it allows administrators to define how to proceed when the package being installed already exists on the system.

/var/sadm/install/admin/default is the default **admin** file delivered with this release. The default file is not writable, so to assign values different from this file, create a new **admin** file. There are no naming restrictions for **admin** files. Name the file when installing a package with the **–a** option of **pkgadd**(1M). If the **–a** option is not used, the default **admin** file is used.

Each entry in the **admin** file is a line that establishes the value of a parameter in the following form:

param=value

Eleven parameters can be defined in an **admin** file. A file is not required to assign values to all eleven parameters. If a value is not assigned, **pkgadd** asks the installer how to proceed.

The eleven parameters and their possible values are shown below except as noted. They may be specified in any order. Any of these parameters can be assigned the value **ask**, which means that if the situation occurs the installer is notified and asked to supply instructions at that time.

basedir Indicates the base directory where relocatable packages are to be installed. If none is specified the installer is prompted with a path, with the default being BASEDIR specified in the **pkginfo** file. The value **default** can be used as an option. **pkgadd** recognizes this setting and installs the package into <*pkginfo BASEDIR*>. The valued may contain **$PKGINST** to indicate a base directory that is to be a function of the package instance.

mail Defines a list of users to whom mail should be sent following installation of a package. If the list is empty, no mail is sent. If the parameter is not present in the **admin** file, the default value of **root** is used. The **ask** value cannot be used with this parameter.

runlevel Indicates resolution if the run level is not correct for the installation or removal of a package. Options are:

- **nocheck** Do not check for run level.
- **quit** Abort installation if run level is not met.

conflict Specifies what to do if an installation expects to overwrite a previously installed file, thus creating a conflict between packages. Options are:

- **nocheck** Do not check for conflict; files in conflict will be overwritten.
- **quit** Abort installation if conflict is detected.

nochange Override installation of conflicting files; they will not be installed.

setuid Checks for executables which will have setuid or setgid bits enabled after installation. Options are:

- **nocheck** Do not check for setuid executables.
- **quit** Abort installation if setuid processes are detected.
- **nochange** Override installation of setuid processes; processes will be installed without setuid bits enabled.

action Determines if action scripts provided by package developers contain possible security impact. Options are:

- **nocheck** Ignore security impact of action scripts.
- **quit** Abort installation if action scripts may have a negative security impact.

partial Checks to see if a version of the package is already partially installed on the system. Options are:

- **nocheck** Do not check for a partially installed package.
- **quit** Abort installation if a partially installed package exists.

instance Determines how to handle installation if a previous version of the package (including a partially installed instance) already exists. Options are:

- **quit** Exit without installing if an instance of the package already exists (does not overwrite existing packages).
- **overwrite** Overwrite an existing package if only one instance exists. If there is more than one instance, but only one has the same architecture, it overwrites that instance. Otherwise, the installer is prompted with existing instances and asked which to overwrite.
- **unique** Do not overwrite an existing instance of a package. Instead, a new instance of the package is created. The new instance will be assigned the next available instance identifier.

idepend Controls resolution if other packages depend on the one to be installed. Options are:

- **nocheck** Do not check package dependencies.
- **quit** Abort installation if package dependencies are not met.

rdepend Controls resolution if other packages depend on the one to be removed. Options are:

- **nocheck** Do not check package dependencies.

quit Abort removal if package dependencies are not met.

space Controls resolution if disk space requirements for package are not met. Options are:

nocheck Do not check space requirements (installation fails if it runs out of space).

quit Abort installation if space requirements are not met.

EXAMPLES

Below is a sample **admin** file.

```
basedir=default
runlevel=quit
conflict=quit
setuid=quit
action=quit
partial=quit
instance=unique
idepend=quit
rdepend=quit
space=quit
```

SEE ALSO

pkgadd(1M)

NOTES

The value **ask** should not be defined in an **admin** file that will be used for non-interactive installation (since by definition, there is no installer interaction). Doing so causes installation to fail when input is needed.

NAME

aliases, addresses, forward – addresses and aliases for sendmail

SYNOPSIS

/etc/mail/aliases
/etc/mail/aliases.dir
/etc/mail/aliases.pag
~/.forward

DESCRIPTION

These files contain mail addresses or aliases, recognized by **sendmail**(1M) for the local host:

/etc/passwd Mail addresses (usernames) of local users.

/etc/aliases Aliases for the local host, in ASCII format. This file can be edited to add, update, or delete local mail aliases.

/etc/aliases. { dir , pag}
The aliasing information from **/etc/aliases**, in binary, **dbm** format for use by **sendmail**(1M). The program **newaliases**(1), which is invoked automatically by **sendmail**(1M), maintains these files.

~/.forward Addresses to which a user's mail is forwarded (see **Automatic Forwarding**, below).

In addition, the YP name services aliases map *mail.aliases* contains addresses and aliases available for use across the network.

Addresses

As distributed, **sendmail**(1M) supports the following types of addresses:

Local Usernames

username

Each local *username* is listed in the local host's **/etc/passwd** file.

Local Filenames

pathname

Messages addressed to the absolute *pathname* of a file are appended to that file.

Commands

| *command*

If the first character of the address is a vertical bar, (**|**), **sendmail**(1M) pipes the message to the standard input of the *command* the bar precedes.

DARPA-standard Addresses

*username***@***domain*

If *domain* does not contain any '**.**' (dots), then it is interpreted as the name of a host in the current domain. Otherwise, the message is passed to a *mailhost* that determines how to get to the specified domain. Domains are divided into subdomains separated by dots, with the top-level domain on the right. Top-level domains include:

.COM Commercial organizations.

.EDU Educational organizations.

.GOV Government organizations.

.MIL Military organizations.

For example, the full address of John Smith could be:

js@jsmachine.Podunk-U.EDU

if he uses the machine named **jsmachine** at Podunk University.

uucp Addresses

. . . [*host***!**]*host***!***username*

These are sometimes mistakenly referred to as ''Usenet'' addresses. **uucp**(1C) provides links to numerous sites throughout the world for the remote copying of files.

Other site-specific forms of addressing can be added by customizing the **sendmail.cf** configuration file. See **sendmail**(1M) for details. Standard addresses are recommended.

Aliases

Local Aliases

/etc/aliases is formatted as a series of lines of the form

*aliasname***:***address*[**,** *address*]

aliasname is the name of the alias or alias group, and *address* is the address of a recipient in the group. Aliases can be nested. That is, an *address* can be the name of another alias group. Because of the way **sendmail**(1M) performs mapping from upper-case to lower-case, an *address* that is the name of another alias group must not contain any upper-case letters.

Lines beginning with white space are treated as continuation lines for the preceding alias. Lines beginning with **#** are comments.

Special Aliases

An alias of the form:

owner-aliasname : *address*

directs error-messages resulting from mail to *aliasname* to *address*, instead of back to the person who sent the message.

An alias of the form:

*aliasname***: :include:***pathname*

with colons as shown, adds the recipients listed in the file *pathname* to the *aliasname* alias. This allows a private list to be maintained separately from the aliases file.

YP Domain Aliases

Normally, the aliases file on the master YP server is used for the *mail.aliases* YP map, which can be made available to every YP client. Thus, the **/etc/mail/aliases*** files on the various hosts in a network will one day be obsolete. Domain-wide aliases should ultimately be resolved into usernames on specific hosts. For example, if the following were in the domain-wide alias file:

jsmith:js@jsmachine

then any YP client could just mail to **jsmith** and not have to remember the machine and username for John Smith. If a YP alias does not resolve to an address with a specific host, then the name of the YP domain is used. There should be an alias of the domain name for a host in this case.

For example, the alias:

jsmith:root

sends mail on a YP client to **root@podunk-u** if the name of the YP domain is **podunk-u**.

Automatic Forwarding

When an alias (or address) is resolved to the name of a user on the local host, **sendmail**(1M) checks for a **~/.forward** file, owned by the intended recipient, in that user's home directory, and with universal read access. This file can contain one or more addresses or aliases as described above, each of which is sent a copy of the user's mail.

Care must be taken to avoid creating addressing loops in the **~/.forward** file. When forwarding mail between machines, be sure that the destination machine does not return the mail to the sender through the operation of any YP aliases. Otherwise, copies of the message may ''bounce.'' Usually, the solution is to change the YP alias to direct mail to the proper destination.

A backslash before a username inhibits further aliasing. For instance, to invoke the **vacation** program, user **js** creates a **~/.forward** file that contains the line:

\js, "|/usr/ucb/vacation js"

so that one copy of the message is sent to the user, and another is piped into the **vacation** program.

FILES

/etc/passwd
/etc/mail/aliases
/etc/mail/sendmail.cf
~/.forward

SEE ALSO

vacation(1), **newaliases**(1), **uucp**(1C), **sendmail**(1M), **dbm**(3B)

NOTES

Because of restrictions in **dbm**(3B), a single alias cannot contain more than about 1000 characters. Nested aliases can be used to circumvent this limit.

NAME

ar – archive file format

SYNOPSIS

```
#include <ar.h>
```

DESCRIPTION

The archive command **ar** is used to combine several files into one. Archives are used mainly as libraries to be searched by the link editor **ld**.

Each archive begins with the archive magic string.

```
#define  ARMAG   "!<arch>\n"        /* magic string */
#define  SARMAG  8                  /* length of magic string */
```

Following the archive magic string are the archive file members. Each file member is preceded by a file member header which is of the following format:

```
#define  ARFMAG   "`\n"         /* header trailer string */

struct  ar_hdr                  /* file member header */
{
   char    ar_name[16];         /* '/' terminated file member name */
   char    ar_date[12];         /* file member date */
   char    ar_uid[6];           /* file member user identification */
   char    ar_gid[6];           /* file member group identification */
   char    ar_mode[8];          /* file member mode (octal) */
   char    ar_size[10];         /* file member size */
   char    ar_fmag[2];          /* header trailer string */
};
```

All information in the file member headers is in printable ASCII. The numeric information contained in the headers is stored as decimal numbers (except for *ar_mode* which is in octal). Thus, if the archive contains printable files, the archive itself is printable.

If the file member name fits, the *ar_name* field contains the name directly, and is terminated by a slash (*/*) and padded with blanks on the right. If the member's name does not fit, *ar_name* contains a slash (*/*) followed by a decimal representation of the name's offset in the archive string table described below.

The *ar_date* field is the modification date of the file at the time of its insertion into the archive. Common format archives can be moved from system to system as long as the portable archive command **ar** is used.

Each archive file member begins on an even byte boundary; a newline is inserted between files if necessary. Nevertheless, the size given reflects the actual size of the file exclusive of padding.

Notice there is no provision for empty areas in an archive file.

Each archive that contains object files (see **a.out**(4)) includes an archive symbol table. This symbol table is used by the link editor **ld** to determine which archive members must be loaded during the link edit process. The archive symbol table (if it exists) is always the first file in the archive (but is never listed) and is automatically created and / or updated by **ar**.

The archive symbol table has a zero length name (that is, **ar_name[0]** is **'/'**), **ar_name[1]==' '**, etc.). All ''words'' in this symbol table have four bytes, using the machine-independent encoding shown below. All machines use the encoding described here for the symbol table, even if the machine's ''natural'' byte order is different.

0x01020304

0	1	2	3
01	02	03	04

The contents of this file are as follows:

1. The number of symbols. Length: 4 bytes.
2. The array of offsets into the archive file. Length: 4 bytes * ''the number of symbols''.
3. The name string table. Length: *ar_size* – 4 bytes * (''the number of symbols'' + 1).

As an example, the following symbol table defines 4 symbols. The archive member at file offset 114 defines *name* and *object*. The archive member at file offset 426 defines and a second version of *name*.

Example Symbol Table

Offset	+0	+1	+2	+3	
0	**4**				4 offset entries
4	**114**				**name**
8	**114**				**object**
12	**426**				**function**
16	**426**				**name**
20	**n**	**a**	**m**	**e**	
24	**\0**	**o**	**b**	**j**	
28	**e**	**c**	**t**	**\0**	
32	**f**	**u**	**n**	**c**	
36	**t**	**i**	**o**	**n**	
40	**\0**	**n**	**a**	**m**	
44	**e**	**\0**			

The string table contains exactly as many null terminated strings as there are elements in the offsets array. Each offset from the array is associated with the corresponding name from the string table (in order). The names in the string table are all the defined global symbols found in the common object files in the archive. Each offset is the location of the archive header for the associated symbol.

If some archive member's name is more than 15 bytes long, a special archive member contains a table of file names, each followed by a slash and a new-line. This string table member, if present, will precede all ''normal'' archive members. The special archive symbol table is not a ''normal'' member, and must be first if it exists. The **ar_name** entry of the string table's member header holds a zero length name **ar_name[0]=='/'**, followed by one trailing slash (**ar_name[1]=='/'**), followed by blanks (**ar_name[2]==' '**, etc.). Offsets into the string table begin at zero. Example *ar_name* values for short and long file names appear below.

Offset	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9
0	**f**	**i**	**l**	**e**	**_**	**n**	**a**	**m**	**e**	**_**
10	**s**	**a**	**m**	**p**	**l**	**e**	**/**	**\n**	**l**	**o**
20	**n**	**g**	**e**	**r**	**f**	**i**	**l**	**e**	**n**	**a**
30	**m**	**e**	**x**	**a**	**m**	**p**	**l**	**e**	**/**	**\n**

Member Name		ar_name
short-name	**short-name/**	Not in string table
file_name_sample	**/0**	Offset 0 in string table
longerfilenamexample	**/18**	Offset 18 in string table

SEE ALSO

ar(1), **ld**(1), **strip**(1), **a.out**(4)

NOTES

strip will remove all archive symbol entries from the header. The archive symbol entries must be restored via the **–ts** options of the **ar** command before the archive can be used with the link editor **ld**.

NAME

archives – device header

DESCRIPTION

```
/* Magic numbers */

#define CMN_ASC  0x070701   /* Cpio Magic Number for –c header */
#define CMN_BIN  070707     /* Cpio Magic Number for Binary header */
#define CMN_BBS  0143561    /* Cpio Magic Number for Byte-Swap header */
#define CMN_CRC  0x070702   /* Cpio Magic Number for CRC header */
#define CMS_ASC  "070701"   /* Cpio Magic String for –c header */
#define CMS_CHR  "070707"   /* Cpio Magic String for odc header */
#define CMS_CRC  "070702"   /* Cpio Magic String for CRC header */
#define CMS_LEN  6          /* Cpio Magic String length */

/* Various header and field lengths */

#define CHRSZ    76         /* –H odc size minus filename field */
#define ASCSZ    110        /* –c and CRC hdr size minus filename field */
#define TARSZ    512        /* TAR hdr size */

#define HNAMLEN  256        /* maximum filename length for binary and
                               odc headers */
#define EXPNLEN  1024       /* maximum filename length for –c and
                               CRC headers */
#define HTIMLEN  2          /* length of modification time field */
#define HSIZLEN  2          /* length of file size field */

/* cpio binary header definition */

struct hdr_cpio {
        short   h_magic,                /* magic number field */
                h_dev;                  /* file system of file */
        ushort  h_ino,                  /* inode of file */
                h_mode,                 /* modes of file */
                h_uid,                  /* uid of file */
                h_gid;                  /* gid of file */
        short   h_nlink,                /* number of links to file */
                h_rdev,                 /* maj/min numbers for special files */
                h_mtime[HTIMLEN],       /* modification time of file */
                h_namesize,             /* length of filename */
                h_filesize[HSIZLEN];    /* size of file */
        char    h_name[HNAMLEN];        /* filename */
};

/* cpio –H odc header format */

struct c_hdr {
        char    c_magic[CMS_LEN],
                c_dev[6],
                c_ino[6],
                c_mode[6],
                c_uid[6],
                c_gid[6],
                c_nlink[6],
```

```
                c_rdev[6],
                c_mtime[11],
                c_namesz[6],
                c_filesz[11],
                c_name[HNAMLEN];
};

/* –c and CRC header format */

struct Exp_cpio_hdr {
        char    E_magic[CMS_LEN],
                E_ino[8],
                E_mode[8],
                E_uid[8],
                E_gid[8],
                E_nlink[8],
                E_mtime[8],
                E_filesize[8],
                E_maj[8],
                E_min[8],
                E_rmaj[8],
                E_rmin[8],
                E_namesize[8],
                E_chksum[8],
                E_name[EXPNLEN];
};

/* Tar header structure and format */
#define TBLOCK      512         /* length of tar header and data blocks */
#define TNAMLEN     100         /* maximum length for tar file names */
#define TMODLEN     8           /* length of mode field */
#define TUIDLEN     8           /* length of uid field */
#define TGIDLEN     8           /* length of gid field */
#define TSIZLEN     12          /* length of size field */
#define TTIMLEN     12          /* length of modification time field */
#define TCRCLEN     8           /* length of header checksum field */

/* tar header definition */

union tblock {
        char dummy[TBLOCK];
        struct header {
                char  t_name[TNAMLEN];          /* name of file */
                char  t_mode[TMODLEN];          /* mode of file */
                char  t_uid[TUIDLEN];           /* uid of file */
                char  t_gid[TGIDLEN];           /* gid of file */
                char  t_size[TSIZLEN];          /* size of file in bytes */
                char  t_mtime[TTIMLEN];         /* modification time of file */
                char  t_chksum[TCRCLEN];        /* checksum of header */
                char  t_typeflag;               /* flag to indicate type of file */
                char  t_linkname[TNAMLEN];      /* file this file is linked with */
                char  t_magic[6];               /* magic string always "ustar" */
```

```
            char  t_version[2];                 /* version strings always "00" */
            char  t_uname[32];                  /* owner of file in ASCII */
            char  t_gname[32];                  /* group of file in ASCII */
            char  t_devmajor[8];                /* major number for special files */
            char  t_devminor[8];                /* minor number for special files */
            char  t_prefix[155];                /* pathname prefix */
      } tbuf;
};

/* volcopy tape label format and structure */

#define VMAGLEN8
#define VVOLLEN6
#define VFILLEN  464

struct volcopy_label {
      char  v_magic[VMAGLEN],
            v_volume[VVOLLEN],
            v_reels,
            v_reel;
      long  v_time,
            v_length,
            v_dens,
            v_reelblks,         /* u370 added field */
            v_blksize,          /* u370 added field */
            v_nblocks;          /* u370 added field */
      char  v_fill[VFILLEN];
      long  v_offset;           /* used with -e and -reel options */
      int   v_type;             /* does tape have nblocks field? */
} ;
```

NAME

asetenv – ASET environment file

SYNOPSIS

/usr/aset/asetenv

DESCRIPTION

The asetenv file is located in **/usr/aset**, the default operating directory of the Automated Security Enhancement Tool (ASET). An alternative working directory can be specified by the administrators through the **aset** –**d** command or the **ASETDIR** environment variable. See **aset**(1M). **asetenv** contains definitions of environment variables for ASET.

There are 2 sections in this file. The first section is labeled *User Configurable Parameters*. It contains, as the label indicates, environment variables that the administrators can modify to customize ASET behavior to suit their specific needs. The second section is labeled *ASET Internal Environment Variables* and should not be changed. The configurable parameters are explained as follows:

TASK This variable defines the list of tasks that **aset** will execute the next time it runs. The available tasks are:

tune	Tighten system files.
usrgrp	Check user/group.
sysconf	Check system configuration file.
env	Check environment.
cklist	Compare system files checklist.
eeprom	Check **eeprom**(1M) parameters.
firewall	Disable forwarding of IP packets.

CKLISTPATH_LOW
CKLISTPATH_MED
CKLISTPATH_HIGH

These variables define the list of directories to be used by **aset** to create a *checklist* file at the *low*, *medium*, and *high* security levels, respectively. Attributes of all the files in the directories defined by these variables will be checked periodically and any changes will be reported by **aset**. Checks performed on these directories are not recursive. **aset** only checks directories explicitly listed in these variables and does not check subdirectories of them.

YPCHECK

This variable is a boolean parameter. It specifies whether **aset** should extend checking (when applicable) on system tables to their NIS equivalents or not. The value **true** enables it while the value **false** disables it.

UID_ALIASES

This variable specifies an alias file for user ID sharing. Normally, **aset** warns about multiple user accounts sharing the same user ID because it is not advisable for accountability reason. Exceptions can be created using an alias file. User ID sharing allowed by the alias file will not be reported by **aset**. See **asetmasters**(4) for the format of the alias file.

PERIODIC_SCHEDULE

This variable specifies the schedule for periodic execution of ASET. It uses the format of **crontab**(1) entries. Briefly speaking, the variable is assigned a string of the following format:

minutes hours day-of-month month day-of-week

Setting this variable does *not* activate the periodic schedule of ASET. To execute ASET periodically, **aset**(1M) must be run with the **–p** option. See **aset**(1M). For example, if **PERIODIC_SCHEDULE** is set to the following, and **aset**(1M) was started with the **–p** option, **aset** will run at 12:00 midnight every day:

0 0 * * *

EXAMPLES

The following is a sample **asetenv** file, showing the settings of the ASET configurable parameters:

```
CKLISTPATH_LOW=/etc:/
CKLISTPATH_MED=$CHECKLISTPATH_LOW:/usr/bin:/usr/ucb
CKLISTPATH_HIGH=$CHECKLISTPATH_MED:/usr/lib:/usr/sbin
YPCHECK=false
UID_ALIASES=/usr/aset/masters/uid_aliases
PERIODIC_SCHEDULE="0 0 * * *"
TASKS="env sysconf usrgrp"
```

When **aset** **–p** is run with this file, **aset** is executed at midnight of every day. The **/** and **/etc** directories are checked at the *low* security level; the **/**, **/etc**, **/usr/bin**, and **/usr/ucb** directories are checked at the *medium* security level; and the **/**, **/etc**, **/usr/bin**, **/usr/lib**, and **/usr/sbin** directories are checked at the *high* security level. Checking of NIS system files is disabled. The **/usr/aset/masters/uid_aliases** file specifies the used IDs available for sharing. The **env**, **sysconf**, and **usrgrp** tasks will be performed, checking the environment variables, various system tables, and the local **passwd** and **group** files.

SEE ALSO

crontab(1), **aset**(1M), **asetmasters**(4)

ASET Administrator Manual

NAME

asetmasters, tune.low, tune.med, tune.high, uid_aliases, cklist.low, cklist.med, cklist.high – ASET master files

SYNOPSIS

/usr/aset/masters/tune.low
/usr/aset/masters/tune.med
/usr/aset/masters/tune.high
/usr/aset/masters/uid_aliases
/usr/aset/masters/cklist.low
/usr/aset/masters/cklist.med
/usr/aset/masters/cklist.high

DESCRIPTION

The **/usr/aset/masters** directory contains several files used by the Automated Security Enhancement Tool (ASET). **/usr/aset** is the default operating directory for ASET. An alternative working directory can be specified by the administrators through the **aset** –**d** command or the **ASETDIR** environment variable. See **aset**(1M).

These files are provided by default to meet the need of most environments. The administrators, however, can edit these files to meet their specific needs. The format and usage of these files are described below.

All the master files allow comments and blank lines to improve readability. Comment lines must start with a leading "#" character.

tune.low
tune.med
tune.high

These files are used by the **tune** task (see **aset**(1M)) to restrict the permission settings for system objects. Each file is used by ASET at the security level indicated by the suffix. Each entry in the files is of the form:

pathname mode owner group type

where

pathname	is the full pathname
mode	is the permission setting
owner	is the owner of the object
group	is the group of the object
type	is the type of the object It can be **symlink** for a symbolic link, **directory** for a directory, or **file** for everything else.

Regular shell wildcard ("*", "?", ...) characters can be used in the *pathname* for multiple references. See **sh**(1). The *mode* is a five-digit number that represents the permission setting. Note that this setting represents a least restrictive value. If the current setting is already more restrictive than the specified value, ASET does not loosen the permission settings.

For example, if *mode* is **00777**, the permission will not be changed, since it is always less restrictive than the current setting.

Names must be used for *owner* and *group* instead of numeric ID's. **?** can be used as a "don't care" character in place of *owner*, *group*, and *type* to prevent ASET from changing the existing values of these parameters.

uid_alias

This file allows user ID's to be shared by multiple user accounts. Normally, ASET discourages such sharing for accountability reason and reports user ID's that are shared. The administrators can, however, define permissible sharing by adding entries to the file. Each entry is of the form:

uid=alias1=alias2=alias3= ...

where

uid is the shared user id

alias? is the user accounts sharing the user ID

For example, if **sync** and **daemon** share the user ID **1**, the corresponding entry is:

1=sync=daemon

cklist.low
cklist.med
cklist.high

These files are used by the **cklist** task (see **aset**(1M)), and are created the first time the task is run at the *low*, *medium*, and *high* levels. When the **cklist** task is run, it compares the specified directory's contents with the appropriate **cklist.***level* file and reports any discrepancies.

EXAMPLES

The following is an example of valid entries for the **tune.low**, **tune.med**, and **tune.high** files:

/bin	**00777**	**root**	**staff**	**symlink**
/etc	**02755**	**root**	**staff**	**directory**
/dev/sd*	**00640**	**root**	**operator**	**file**

SEE ALSO

aset(1M), **asetenv**(4)

ASET Administrator Manual

NAME

audit.log – audit trail file

SYNOPSIS

#include <bsm/audit.h>
#include <bsm/audit_record.h>

AVAILABILITY

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

DESCRIPTION

audit.log files are the depository for audit records stored locally or on an audit server. These files are kept in directories named in the file **audit_control**(4). They are named to reflect the time they are created and are, when possible, renamed to reflect the time they are closed as well. The name takes the form

*yyyymmddhhmmss***.not_terminated.***hostname*

when open or if the **auditd**(1M) terminated ungracefully, and the form

*yyyymmddhhmmss***.***yyyymmddhhmmss***.***hostname*

when properly closed. **yyyy** is the year, **mm** the month, **dd** day in the month, **hh** hour in the day, **mm** minute in the hour, and **ss** second in the minute. All fields are of fixed width.

The **audit.log** file begins with a standalone **file token** and typically ends with one also. The beginning **file token** records the pathname of the previous audit file, while the ending **file token** records the pathname of the next audit file. If the file name is NULL the appropriate path was unavailable.

The **audit.log** files contains audit records. Each audit record is made up of *audit tokens*. Each record contains a header token followed by various data tokens. Depending on the audit policy in place by **auditon**(2), optional other tokens such as trailers or sequences may be included.

The tokens are defined as follows:

The **file** token consists of:

token ID	char
seconds of time	u_int
milliseconds of time	u_int
file name length	short
file pathname	null terminated string

The **header** token consists of:

token ID	char
record byte count	u_long
version #	char (1)
event type	u_short
event modifier	u_short
seconds of time	u_int
milliseconds of time	u_int

The **trailer** token consists of:

token ID	char
trailer magic number	u_short
record byte count	u_long

The **arbitrary data** token is defined:

token ID	char
how to print	char
basic unit	char
unit count	char
data items	*depends on basic unit*

The **in_addr** token consists of:

token ID	char
internet address	long

The **ip** token consists of:

token ID	char
version and ihl	char
type of service	char
length	short
id	u_short
offset	u_short
ttl	char
protocol	char
checksum	u_short
source address	long
destination address	long

The **iport** token consists of:

token ID	char
port address	short

The **opaque** token consists of:

token ID	char
size	short
data	char, *size* chars

The **path** token consists of:

token ID	char
path length	short
path	null terminated string

The **process** token consists of:

token ID	char
auid	u_long
euid	u_long
egid	u_long
ruid	u_long
rgid	u_long

pid	u_long
sid	u_long
terminal ID	u_long (port ID)
	u_long (machine ID)

The **return** token consists of:

token ID	char
error number	char
return value	u_int

The **subject** token consists of:

token ID	char
auid	u_long
euid	u_long
egid	u_long
ruid	u_long
rgid	u_long
pid	u_long
sid	u_long
terminal ID	u_long (port ID)
	u_long (machine ID)

The **System V IPC** token consists of:

token ID	char
object ID type	char
object ID	long

The **text** token consists of:

token ID	char
text length	short
text	null terminated string

The **attribute** token consists of:

token ID	char
mode	u_long
uid	u_long
gid	u_long
file system id	long
node id	long
device	u_long

The **groups** token consists of:

token ID	char
number	short
group list	long, *size* chars

The **System V IPC permission** token consists of:

token ID	char
uid	u_long
gid	u_long

cuid	u_long
cgid	u_long
mode	u_long
seq	u_long
key	long

The **arg** token consists of:

token ID	char
argument #	char
argument value	long
string length	short
text	null terminated string

The **exec_args** token consists of:

token ID	char
count	short
text	*count* null terminated string(s)

The **exec_env** token consists of:

token ID	char
count	short
text	*count* null terminated string(s)

The **exit** token consists of:

token ID	char
status	long
return value	long

The **socket** token consists of:

token ID	char
socket type	short
local port	short
local Internet address	long
remote port	short
remote Internet address	long

The **seq** token consists of:

token ID	char
sequence number	long

SEE ALSO **audit**(1M), **auditd**(1M), **bsmconv**(1M), **audit**(2), **auditon**(2), **audit_control**(4)

NOTES Each token is generally written using the **au_to**(3) family of function calls.

NAME

audit_class – audit class definitions

SYNOPSIS

/etc/security/audit_class

AVAILABILITY

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

DESCRIPTION

/etc/security/audit_class is an ASCII system file that stores class definitions. Programs use the **getauclassent**(3) routines to access this information.

The fields for each class entry are separated by colons. Each class entry is a bitmap and is separated from each other by a newline.

Each entry in the audit_class file has the form:

mask:*name*:*description*

The fields are defined as follows:

mask	The class mask.
name	The class name.
description	The description of the class.

The classes are now user-configurable. Each class is represented as a bit in the class mask which is an unsigned integer. Thus, there are 32 different classes available, plus two meta-classes -- **all** and **no**.

all represents a conjunction of all allowed classes, and is provided as a shorthand method of specifying all classes.

no is the "invalid" class, and any event mapped solely to this class will not be audited. (Turning auditing on to the **all** meta class will NOT cause events mapped solely to the **no** class to be written to the audit trail.)

EXAMPLES

Here is a sample of a **audit_class** file:

0x00000000:no:invalid class
0x00000001:fr:file read
0x00000002:fw:file write
0x00000004:fa:file attribute access
0x00000008:fm:file attribute modify
0x00000010:fc:file create
0x00000020:fd:file delete
0x00000040:cl:file close
0xffffffff:all:all classes

FILES

/etc/security/audit_class

SEE ALSO **bsmconv**(1M), **getauclassent**(3), **audit_event**(4)

NOTES It is possible to deliberately turn on the **no** class in the kernel, in which case the audit trail will be flooded with records for the audit event AUE_NULL.

NAME

audit_control – control information for system audit daemon

SYNOPSIS

/etc/security/audit_control

AVAILABILITY

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

DESCRIPTION

The **audit_control** file contains audit control information used by **auditd**(1M). Each line consists of a title and a string, separated by a colon. There are no restrictions on the order of lines in the file, although some lines must appear only once. A line beginning with '**#**' is a comment.

Directory definition lines list the directories to be used when creating audit files, in the order in which they are to be used. The format of a directory line is:

dir: *directory-name*

directory-name is where the audit files will be created. Any valid writable directory can be specified.

The following configuration is recommended:

/etc/security/audit/*server* / files

where *server* is the name of a central machine, since audit files belonging to different servers are usually stored in separate subdirectories of a single audit directory. The naming convention normally has *server* be a directory on a server machine, and all clients mount **/etc/security/audit/***server* at the same location in their local file systems. If the same server exports several different file systems for auditing, their *server* names will, of course, be different.

There are several other ways for audit data to be arranged: some sites may have needs more in line with storing each host's audit data in separate subdirectories. The audit structure used will depend on each individual site.

The audit threshold line specifies the percentage of free space that must be present in the file system containing the current audit file. The format of the threshold line is:

minfree: *percentage*

where *percentage* is indicates the amount of free space required. If free space falls below this threshold, the audit daemon **auditd**(1M) invokes the shell script **audit_warn**(1M). If no threshold is specified, the default is 0%.

The audit flags line specifies the default system audit value. This value is combined with the user audit value read from **audit_user**(4) to form the process audit state. The user audit value overrides the system audit value. The format of a flags line is:

flags:*audit-flags*

where *audit-flags* specifies which event classes are to be audited. The character string representation of *audit-flags* contains a series of flag names, each one identifying a single audit class, separated by commas. A name preceded by '–' means that the class should be audited for failure only; successful attempts are not audited. A name preceded by '+' means that the class should be audited for success only; failing attempts are not audited. Without a prefix, the name indicates that the class is to be audited for both successes and failures. The special string **all** indicates that all events should be audited; –**all** indicates that all failed attempts are to be audited, and **+all** all successful attempts. The prefixes **^**, **^–**, and **^+** turn off flags specified earlier in the string (**^–** and **^+** for failing and successful attempts, **^** for both). They are typically used to reset flags.

The non-attributable flags line is similar to the flags line, but this one contain the audit flags that define what classes of events are audited when an action cannot be attributed to a specific user. The format of a **naflags** line is:

naflags: *audit-flags*

The flags are separated by commas, with no spaces.

The following table lists the predefined audit classes:

short name	long name	short description
no	no_class	null value for turning off event preselection
fr	file_read	Read of data, open for reading, etc.
fw	file_write	Write of data, open for writing, etc.
fa	file_attr_acc	Access of object attributes: stat, pathconf, etc.
fm	file_attr_mod	Change of object attributes: chown, flock, etc.
fc	file_creation	Creation of object
fd	file_deletion	Deletion of object
cl	file_close	close(2) system call
pc	process	Process operations: fork, exec, exit, etc.
nt	network	Network events: bind, connect, accept, etc.
ip	ipc	System V IPC operations
na	non_attrib	non-attributable events
ad	administrative	administrative actions: mount, exportfs, etc.
lo	login_logout	Login and logout events
ap	application	Application auditing
io	ioctl	ioctl(2) system call
ex	exec	exec(2) system call
ot	other	Everything else
all	all	All flags set

Note that the classes are configurable, see **audit_class**(4).

EXAMPLES

Here is a sample **/etc/security/audit_control** file for the machine eggplant:

```
dir: /etc/security/jedgar/eggplant
dir: /etc/security/jedgar.aux/eggplant
#
# Last-ditch audit file system when jedgar fills up.
#
dir: /etc/security/global/eggplant
minfree: 20
flags: lo,ad,-all,^-fm
naflags: lo,ad
```

This identifies server **jedgar** with two file systems normally used for audit data, another server **global** used only when **jedgar** fills up or breaks, and specifies that the warning script is run when the file systems are 80% filled. It also specifies that all logins, administrative operations are to be audited (whether or not they succeed), and that failures of all types except failures to access object attributes are to be audited.

FILES

/etc/security/audit_control
/etc/security/audit_warn
/etc/security/audit/*/*/*
/etc/security/audit_user

SEE ALSO

audit(1M), **auditd**(1M), **audit_warn**(1M), **bsmconv**(1M), **audit**(2), **getfauditflags**(3), **audit.log**(4), **audit_class**(4), **audit_user**(4)

NAME

audit_data – current information on audit daemon

SYNOPSIS

/etc/security/audit_data

AVAILABILITY

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

DESCRIPTION

The **audit_data** file contains information about the audit daemon. The file contains the process ID of the audit daemon, and the pathname of the current audit log file. The format of the file is:

<*pid*>:<*pathname*>

Where *pid* is the process ID for the audit daemon, and *pathname* is the full pathname for the current audit log file.

EXAMPLES

64:/etc/security/audit/server1/19930506081249.19930506230945.bongos

FILES

/etc/security/audit_data

SEE ALSO

audit(1M), **auditd**(1M), **bsmconv**(1M), **audit**(2), **audit.log**(4)

NAME

audit_event – audit event definition and class mapping

SYNOPSIS

/etc/security/audit_event

AVAILABILITY

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

DESCRIPTION

/etc/security/audit_event is an ASCII system file that stores event definitions and specifies the event to class mappings. Programs use the **getauevent**(3) routines to access this information.

The fields for each event entry are separated by colons. Each event is separated from the next by a newline.

Each entry in the audit_event file has the form:

number:*name*:*description*:*flags*

The fields are defined as follows:

number	The event number.
name	The event name.
description	The description of the event.
flags	Flags specifying classes to which the event is mapped.

EXAMPLES

Here is a sample of the **audit_event** file entries:

```
7:AUE_EXEC:exec(2):pc,ex
79:AUE_OPEN_WTC:open(2) - write,creat,trunc:fc,fd,fw
6152:AUE_login:login - success or failure:lo
6153:AUE_logout:logout:lo
6154:AUE_telnet:login - through telnet:lo
6155:AUE_rlogin:login - through rlogin:lo
```

FILES

/etc/security/audit_event

SEE ALSO

bsmconv(1M), **getauevent**(3), **audit_control**(4)

NAME

audit_user – per-user auditing data file

SYNOPSIS

/etc/security/audit_user

AVAILABILITY

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

DESCRIPTION

audit_user is an access-restricted ASCII system file that stores per-user auditing preselection data. Programs use the **getauusernam**(3) routines to access this information.

The fields for each user entry are separated by colons. Each user is separated from the next by a newline. **audit_user** does not have general read permission.

Each entry in the audit_user file has the form:

username:*always-audit-flags*:*never-audit-flags*

The fields are defined as follows:

username	The user's login name.
always-audit-flags	Flags specifying event classes to *always* audit.
never-audit-flags	Flags specifying event classes to *never* audit.

EXAMPLES

Here is a sample **audit_user** file:

```
other:lo,ad:io,cl
fred:lo,ex,+fc,-fr,-fa:io,cl
ethyl:lo,ex,nt:io,cl
```

FILES

/etc/security/audit_user
/etc/passwd

SEE ALSO

bsmconv(1M), **getauusernam**(3), **audit_control**(4), **passwd**(4)

NAME

bootparams – boot parameter data base

SYNOPSIS

/etc/bootparams

DESCRIPTION

The **/etc/bootparams** file contains a list of client entries that diskless clients use for booting. Diskless booting clients retrieve this information by issuing requests to a server running the **rpc.bootparamd**(1M) program. The **/etc/bootparams** file may be used in conjunction with or in place of other sources for the **bootparams** information. See **nsswitch.conf**(4).

For each client the file contains an entry with the client's name and a list of boot parameter values for that client. Each entry should have the form:

clientname identifier-pathname-specifier ...

The first item of each entry is the host name of the diskless client. This is followed by one or more whitespace characters and a series of identifier-pathname-specifiers separated by whitespace characters.

Each identifier-pathname-specifier has the form:

identifier=*server*:*pathname*

where *identifier* is a key that is used by diskless clients to identify a file or filesystem, *server* is the name of the server that will provide the file or filesystem to the diskless client, and *pathname* is the path to the exported file or filesystem on the specified server. The equal sign ('=') and colon (':') characters are used in the indicated positions. There should not be any whitespace within an identifier-pathname-specifier.

An entry may be split across multiple lines of the file. The backslash ('\') character should be used as the last character of a line to signify that the entry continues on the next line. The line may only be split in places where whitespace is allowed in the entry.

The asterisk ('*') character may be used as a "wildcard" in place of the client name in a single entry. That entry will apply to all clients for whom there is not an entry that specifically names them.

EXAMPLES

Here is an example of an entry in the **/etc/bootparams** file:

```
client1   root=server1:/export/client1/root \
        swap=server1:/export/client1/swap
```

FILES

/etc/bootparams

SEE ALSO

rpc.bootparamd(1M), **nsswitch.conf**(4)

x86 only

rpld(1M)

NOTES

Solaris diskless clients use the identifiers "root", "swap", and "dump" to look up the pathnames for the root filesystem, a swap area, and a dump area, respectively. These are the only identifiers meaningful for SPARC diskless booting clients.

For x86 booting clients, the additional keyword identifiers "numbootfiles," "bootfile," and "bootaddr" are used (see **rpld**(1M)).

NAME

cdtoc – CD-ROM table of contents file

DESCRIPTION

The table of contents file, **.cdtoc**, is an ASCII file that describes the contents of a CD-ROM or other software distribution media. It resides in the top-level directory of the file system on a slice of a CD-ROM. It is independent of file system format, that is, the file system on the slice can be either *ufs* or *hsfs*.

Each entry in the **.cdtoc** file is a line that establishes the value of a parameter in the following form:

PARAM=*value*

Blank lines and comments (lines preceded by a pound-sign, ''#'') are also allowed in the file. Parameters are grouped by product, with the beginning of a product defined by a line of the form:

PRODNAME=*value*

Each product is expected to consist of one or more software packages that are stored together in a subdirectory on the distribution media. There can be any number of products described within the file. There is no required order in which the parameters must be specified, except that the parameters must be grouped by product and the *PRODNAME* parameter must appear first in the list of parameters for each product specified. Each parameter is described below. All of the parameters are required for each product.

PRODNAME The full name of the product. This must be unique within the **.cdtoc** file and is preferably unique across all possible products. This value may contain white space. The length of this value is limited to 256 ASCII characters; other restrictions may apply (see below).

PRODVERS The version of the product. The value can contain any combination of letters, numbers, or other characters. This value may contain white space. The length of this value is limited to 256 ASCII characters; other restrictions may apply (see below).

PRODDIR The name of the top-level directory containing the product. This name should be relative to the top-level directory of the distribution media, for example, *Solaris_2.0*. The number of path components in the name is limited only by the system's maximum path name length, which is 1024 ASCII characters. Any single component is limited to 256 ASCII characters. This value cannot contain white space.

The lengths of the values of *PRODNAME* and *PRODVERS* are further constrained by the fact that the initial install programs and **swmtool**(1M) concatenate these values to produce the full product name. **swmtool** concatenates the two values (inserting a space) to produce the name displayed in its software selection menu, for example, *Solaris 2.0*. For unbundled products the combined length of the values of *PRODNAME* and *PRODVERS* must not exceed 256 ASCII characters.

During installation of the bundled OS release, directories for diskless and dataless client *usr* and *kvm* file systems are created by constructing names derived from a concatenation of the values of *PRODNAME*, *PRODVERS*, and client architecture, for example,

/export/exec/kvm/Solaris_2.0_sparc.sun4c/usr/kvm. The length of the component containing the product name and version must not exceed 256 ASCII characters. Thus, for products corresponding to bundled OS releases (for example, Solaris 2.0), the values of *PRODNAME* and *PRODVERS* are effectively restricted to lengths much less than 256.

The initial installation programs, **swm** and **swmtool**, use the value of the *PRODDIR* macro in the **.cdtoc** file to indicate where packages can be found.

EXAMPLES

Here is a sample **.cdtoc** file:

```
#
# .cdtoc file -- Online product family CD
#
PRODNAME=Online DiskSuite
PRODVERS=2.0
PRODDIR=Online_DiskSuite_2.0
#
PRODNAME=Online Backup
PRODVERS=2.0
PRODDIR=Online_Backup_2.0
```

This example corresponds to the following directory layout on a CD-ROM partition:

```
/.cdtoc
/Online_DiskSuite_2.0
        ./SUNWmddr.c
        ./SUNWmddr.m
        ./SUNWmddu
/Online_Backup_2.0
        ./SUNWhsm
```

The bundled release of Solaris 2.2 includes the following **.cdtoc** file:

```
PRODNAME=Solaris
PRODVERS=2.2
PRODDIR=Solaris_2.2
```

This file corresponds to the following directory layout on slice 0 of the Solaris 2.0 product CD:

```
/.cdtoc
/Solaris_2.2
        ./SUNWaccr
        ./SUNWaccu
        ./SUNWadmap
        .
        .
        .
        ./SUNWutool
```

SEE ALSO **swmtool**(1M), **clustertoc**(4), **packagetoc**(4), **pkginfo**(4)

NAME

clustertoc – cluster table of contents description file

DESCRIPTION

The cluster table of contents file, **.clustertoc,** is an ASCII file that describes a hierarchical view of a software product. A **.clustertoc** file is required for the base OS product. The file resides in the top-level directory containing the product.

The hierarchy described by **.clustertoc** can be of arbitrary depth, although the initial system installation programs assume that it has three levels. The hierarchy is described bottom-up, with the packages described in **.packagetoc** at the lowest layer. The next layer is the *cluster* layer which collects packages into functional units. The highest layer is the *meta-cluster* layer which collects packages and clusters together into typical configurations.

The hierarchy exists to facilitate the selection or deselection of software for installation at varying levels of granualarity. Interacting at the package level gives the finest level of control over what software is to be installed.

Each entry in the **.clustertoc** file is a line that establishes the value of a parameter in the following form:

PARAM=*value*

A line starting with a pound-sign, ''#'', is considered a comment and is ignored.

Parameters are grouped by cluster or meta-cluster. The start of a cluster description is defined by a line of the form:

CLUSTER=*value*

The start of a meta-cluster description is defined by a line of the form:

METACLUSTER=*value*

There is no order implied or assumed for specifying the parameters for a (meta-)cluster with the exception of the *CLUSTER* or *METACLUSTER* parameter, which must appear first and the *END* parameter which must appear last.

Each parameter is described below. All of the parameters are mandatory.

CLUSTER The cluster identifier (for example, SUNWCacc). The identifier specified must be unique within the package and cluster identifier namespace defined by a product's **.packagetoc** and **.clustertoc** files. The identifiers used are subject to the same constraints as those for package identifiers. These constraints are (from **pkginfo**(4)):

''All characters in the abbreviation must be alphanumeric and the first may not be numeric. The abbreviation is limited to a maximum length of nine characters. **install**, **new**, and **all** are reserved abbreviations.''

A cluster must be described before another cluster or meta-cluster may refer to it.

METACLUSTER The metacluster identifier (for example, *SUNWCprog*). The identifier specified must be unique within the package and cluster identifier namespace defined by a product's **.packagetoc** and **.clustertoc** files. The identifiers used are subject to the same constraints as those for package identifiers. These constraints are (from **pkginfo**(4)):

''All characters in the abbreviation must be alphanumeric and the first may not be numeric. The abbreviation is limited to a maximum length of nine characters. **install**, **new**, and **all** are reserved abbreviations.''

Meta-clusters *cannot* contain references to other meta-clusters.

NAME The full name of the (meta-)cluster. The length of the name string supplied may not exceed 256 characters.

VENDOR The name of the (meta-)cluster's vendor. The length of the vendor string supplied may not exceed 256 characters.

VERSION The version of the (meta-)cluster. The length of the version string supplied may not exceed 256 characters.

DESC An informative textual description of the (meta-)cluster's contents. The length of the description supplied may not exceed 256 characters. The text should contain no newlines.

SUNW_CSRMEMBER
Indicates that the package or cluster is a part of the (meta-) cluster currently being described. The value specified is the identifier of the package or cluster. There may be an arbitrary number of *SUNW_CSRMEMBER* parameters per (meta-)cluster.

EXAMPLES

The following is an example of a cluster description in a **.clustertoc** file.

```
# ident "@(#)clustertoc.4 1.2 93/02/24"
CLUSTER=SUNWCacc
NAME=System Accounting
DESC=System accounting utilities
VENDOR=Sun Microsystems, Inc.
VERSION=7.2
SUNW_CSRMEMBER=SUNWaccr
SUNW_CSRMEMBER=SUNWaccu
END
```

The following is an example of a meta-cluster description in a **.clustertoc** file.

```
# required meta-cluster description for Solaris 2.0 FCS
METACLUSTER=SUNWCreq
NAME=Core System Support
DESC=A pre-defined software configuration consisting of the minimum
required software for a standalone, non-networked workstation.
VENDOR=Sun Microsystems, Inc.
VERSION=2.0
SUNW_CSRMEMBER=SUNWadmr
SUNW_CSRMEMBER=SUNWcar
SUNW_CSRMEMBER=SUNWCcs
SUNW_CSRMEMBER=SUNWCcg6
SUNW_CSRMEMBER=SUNWCdfb
SUNW_CSRMEMBER=SUNWkvm
SUNW_CSRMEMBER=SUNWCnis
SUNW_CSRMEMBER=SUNWowdv
SUNW_CSRMEMBER=SUNWter
END
```

SEE ALSO

cdtoc(4), **order**(4), **packagetoc**(4), **pkginfo**(4)

NOTES

The current implementation of the initial system installation programs depend on the **.clustertoc** describing three required meta-clusters for the base OS product:

SUNWCall	contains all of the software packages in the OS distribution.
SUNWCuser	contains the typical software packages for an end-user of the OS distribution.
SUNWCreq	contains the bare-minimum packages required to boot and configure the OS to the point of running a multi-user shell.

NAME

compver – compatible versions file

DESCRIPTION

compver is an ASCII file used to specify previous versions of the associated package which are upward compatible. It is created by a package developer.

Each line of the file specifies a previous version of the associated package with which the current version is backward compatible.

Since some packages may require installation of a specific version of another software package, compatibility information is extremely crucial. Consider, for example, a package called "A" which requires version "1.0" of application "B" as a prerequisite for installation. If the customer installing "A" has a newer version of "B" (version 1.3), the **compver** file for "B" must indicate that "1.3" is compatible with version "1.0" in order for the customer to install package "A".

EXAMPLES

A sample **compver** file is shown below.

Version 1.3
Version 1.0

NOTES

The comparison of the version string disregards white space and tabs. It is performed on a word-by-word basis. Thus "Version 1.3" and "Version 1.3" would be considered the same.

NAME

copyright – copyright information file

DESCRIPTION

copyright is an ASCII file used to provide a copyright notice for a package. The text may be in any format. The full file contents (including comment lines) is displayed on the terminal at the time of package installation.

NAME

core – core image file

DESCRIPTION

The operating system writes out a core image of a process when it is terminated due to the receipt of some signals. The core image is called **core** and is written in the process's working directory (provided it can be; normal access controls apply). A process with an effective user ID different from the real user ID will not produce a core image.

The core file contains all the process information pertinent to debugging: contents of hardware registers, process status and process data. The format of a core file is object file specific.

For ELF executable programs (see **a.out**(4)), the core file generated is also an ELF file, containing ELF program and file headers. The **e_type** field in the file header has type **ET_CORE**. The program header contains an entry for every loadable and writeable segment that was part of the process address space, including shared library segments. The contents of the segments themselves are also part of the core image.

The program header of an ELF core file also contains a **NOTE** segment. This segment may contain the following entries. Each has entry name "**CORE**" and presents the contents of a system structure:

prstatus_t
: The entry containing this structure has a **NOTE** type of 1. This structure contains things of interest to a debugger from the operating system's u-area, such as the general registers, signal dispositions, state, reason for stopping, process ID and so forth. The **prstatus_t** structure is defined in **<sys/procfs.h>**.

prfpregset_t
: This entry is present only if the process used the floating-point hardware. It has a **NOTE** type of 2 and contains the floating-point registers. The **prfpregset_t** structure is defined in **<sys/procfs.h>**.

prpsinfo_t
: The entry containing this structure has a **NOTE** type of 3. It contains information of interest to the **ps**(1) command, such as process status, cpu usage, "nice" value, controlling terminal, user ID, process ID, the name of the executable and so forth. The **prpsinfo_t** structure is defined in **<sys/procfs.h>**.

The size of the core file created by a process may be controlled by the user (see **getrlimit**(2)).

SEE ALSO

adb(1), **gcore**(1), **crash**(1M), **getrlimit**(2), **setuid**(2), **elf**(3E), **a.out**(4), **proc**(4), **signal**(5)
ANSI C Programmer's Guide

NAME

default_fs, fs – specify the default file system type for local or remote file systems

DESCRIPTION

When file system administration commands have both specific and generic components (for example, **fsck**(1M)), the file system type must be specified. If it is not explicitly specified using the –**F** *FSType* command line option, the generic command looks in **/etc/vfstab** in order to determine the file system type, using the supplied raw or block device or mount point. If the file system type can not be determined by searching **/etc/vfstab**, the command will use the default file system type specified in either **/etc/default/fs** or **/etc/dfs/dfstypes**, depending on whether the file system is local or remote.

The default local file system type is specified in **/etc/default/fs** by a line of the form **LOCAL=***fstype* (for example, **LOCAL=ufs**). The default remote file system type is determined by the first entry in the **/etc/dfs/fstypes** file.

File system administration commands will determine whether the file system is local or remote by examining the specified device name. If the device name starts with ‘‘/’’ (slash), it is considered to be local; otherwise it is remote.

The default file system types can be changed by editing the default files with a text editor.

FILES

/etc/vfstab	list of default parameters for each file system
/etc/default/fs	the default local file system type
/etc/dfs/fstypes	the default remote file system type

SEE ALSO

fstypes(4), **vfstab**(4)

NAME

depend – software dependencies file

DESCRIPTION

depend is an ASCII file used to specify information concerning software dependencies for a particular package. The file is created by a software developer.

Each entry in the **depend** file describes a single software package. The instance of the package is described after the entry line by giving the package architecture and/or version. The format of each entry and subsequent instance definition is:

type pkg name
(arch)version
(arch)version
...

The fields are:

type Defines the dependency type. Must be one of the following characters:

- P Indicates a prerequisite for installation, for example, the referenced package or versions must be installed.
- I Implies that the existence of the indicated package or version is incompatible.
- R Indicates a reverse dependency. Instead of defining the package's own dependencies, this designates that another package depends on this one. This type should be used only when an old package does not have a **depend** file but it relies on the newer package nonetheless. Therefore, the present package should not be removed if the designated old package is still on the system since, if it is removed, the old package will no longer work.

pkg Indicates the package abbreviation.

name Specifies the full package name.

(arch)version Specifies a particular instance of the software. A version name cannot begin with a left parenthesis. The instance specifications, both *arch* and *version*, are completely optional but must each begin on a new line that begins with white space. A null version set equates to any version of the indicated package.

EXAMPLES

Here is a sample **depend** file:

```
#ident  "@(#)pkg.compat:depend  1.1"
P  nsu          Networking Support Utilities
P  inet         Internet Utilities
P  sys          System Header Files
P  src_compat   Source Compatibility Files
```

NAME

device.cfinfo – devconfig configuration files

SYNOPSIS

device.cfinfo

AVAILABILITY

x86

DESCRIPTION

device.cfinfo files pass information about device configuration to the **devconfig**(1M) program. They allow **devconfig**(1M) to provide the user with valid ranges for device attributes.

devconfig(1M) associates a device with its **cfinfo** file by name. For example, the device **logi** for the Logitec Bus Mouse has the **devconfig**(1M) configuration file **logi.cfinfo** associated with it in the **DEVCONFIGHOME** directory. **DEVCONFIGHOME is /usr/lib/devconfig** by default and may be set in the user's environment.

Below is a yaccish grammar of a **cfinfo** file:

```
cfinfo_file:        cfinfo_devspec EOF
                    ;

cfinfo_devspec:     cfinfo_spec_list SEMICOLON
                    ;

cfinfo_spec_list:   cfinfo_spec |
                    cfinfo_spec_list cfinfo_spec
                    ;

cfinfo_spec:        comment |
                    attr_value_pair NEWLINE
                    ;

comment:            POUNDSIGN |
                    POUNDSIGN STRING
                    ;

attr_value_pair:    ATTR_NAME EQUALS STRING |
                    ATTR_OWNAME EQUALS STRING
                    ATTR_TITLE EQUALS STRING |
                    ATTR_CATEGORY EQUALS STRING |
                    ATTR_INSTANCE EQUALS STRING |
                    ATTR_CLASS EQUALS STRING |
                    ATTR_TYPE EQUALS STRING |
                    ATTR_REAL EQUALS STRING |
                    ATTR_AUTO EQUALS STRING |
                    NAME EQUALS value_spec_string
                    ;
```

```
value_spec_string:  QUOTE value_spec QUOTE
                    ;

value_spec:         value_type COMMA value_list
                    ;

value_type:         | /* EMPTY */
                    TYPE_NUMERIC |
                    TYPE_STRING |
                    TYPE_VAR
                    ;

value_list:         integer_value_list |
                    string_value_list
                    ;

integer_value_list: INTEGER |
                    INTEGER COLON INTEGER |
                    INTEGER COMMA integer_value_list
                    ;

string_value_list:  STRING |
                    STRING COMMA string_value_list
                    ;
ATTR_NAME       name        # device name specified in driver.conf
ATTR_CLASS      class       # device class specified in driver.conf
ATTR_TYPE       type        # device type specified in OWconfig
ATTR_OWNAME     __owname__  # device name specified in OWconfig
ATTR_TITLE      __title__   # device title displayed by devconfig
ATTR_CATEGORY   __category__ # device category
ATTR_INSTANCE   __instance__ # device unit
ATTR_REAL       __real__    # attributes to write to driver.conf
ATTR_AUTO       __auto__    # self-identifying device attribute
TYPE_NUMERIC    numeric     # precedes an integer value list
TYPE_STRING     string      # precedes a string values list
TYPE_VAR        var         # precedes a variable specification
```

The first value in a **value_list** is the default value picked by **devconfig**(1M) for the attribute. An attribute name of the form *__name__* is used internally by **devconfig**(1M). Number ranges are specified as *n1:n2*. An internal attribute of the type *var* specifies a configurable portion of a real attribute. (See examples below.) Certain internal attributes have an expanded form when displayed. These attributes are listed in the file **abbreviations** in **DEVCONFIGHOME**. The file abbreviations also includes a list of name mappings for certain category names. If the *__real__* attribute is present, only the attribute names it specifies are written to a driver.conf file. Otherwise all non-internal attributes are written.

EXAMPLES

Here is the device configuration file **logi.cfinfo** for the LOGITECH bus mouse. The driver configuration file for this device is called **logi.conf**.

```
name="logi"
                __ownname__="pointer:0"
                __title__="Logitec bus mouse"

                __category__="pointer"

                class="sysbus"
                type="LOGI-B"
                buttons="var,__nbuttons__"
                __nbuttons__="numeric,2:3"
                dev="/dev/logi"

                intr="numeric,1","var,__irq__"
                __irq__="numeric,2:5"

                __real__="name","class","intr"
                ;
```

The driver name for the LOGITECH Bus Mouse is **logi**. The device name in **OWconfig** (see the *OpenWindows Reference Manual*) is **pointer:0**. The device category is **pointer**; the device category is displayed as **pointing devices** however since there is a category mapping for **pointer** in the abbreviations file. The device class is **sysbus** as specified in the file /kernel/drv/classes. A device of class **owin** does not have a device driver associated with it. The device IPL is 1. The device IRQ is substituted by the variable **__irq__** and has a range of 2 to 5. A name mapping for **__irq__** exists in abbreviations and so **__irq__** is displayed as **Interrupt (IRQ):**. The device attributes written to logi.conf are **name**, **class** and **intr** as specified by the **__real__**" entry.

The resulting entry in **logi.conf** is:

```
name="logi" class="sysbus" intr=1,2;
```

The resulting entry in OWconfig is:

```
type="LOGI-B" buttons=3 dev="/dev/logi" class="owin" name="pointer:0";
```

Here is an example of a self-identifying device.

```
name="lp"
                __title__="Parallel printer port"
                __category__="lp"

                class="sysbus"

                __auto__="string,true"
                ;
```

The driver for the parallel port automatically identifies it, and **devconfig**(1M) treats this device as self-identifying.

FILES

abbreviations

SEE ALSO

devconfig(1M), **driver.conf**(4),
OpenWindows Reference Manual

NAME

device_allocate – device_allocate file

SYNOPSIS

/etc/security/device_allocate

AVAILABILITY

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

DESCRIPTION

The **device_allocate** file contains mandatory access control information about each physical device. Each device is represented by a one line entry of the form:

device-name;*device-type*;reserved;reserved;*alloc*;*device-exec*

where

device-name	This is an arbitrary ASCII string naming the physical device. This field contains no embedded white space or non-printable characters.
device-type	This is an arbitrary ASCII string naming the generic device type. This field identifies and groups together devices of like type. This field contains no embedded white space or non-printable characters.
reserved	This field is reserved for future use.
reserved	This field is reserved for future use.
alloc	This field contains an arbitrary string which controls whether or not a device is allocatable. If the field contains only an asterisk (*), the device is *not* allocatable. Otherwise, the device may be allocated and deallocated in the normal fashion.
device-exec	This is the physical device's data purge program to be run any time the device is acted on by **allocate**(1M). This is to ensure that all usable data is purged from the physical device before it is reused. This field contains the filename of a program in **/etc/security/lib** or the full pathname of a cleanup script provided by the system administrator.

The **device_allocate** file is an ASCII file that resides in the **/etc/security** directory.

Lines in **device_allocate** can end with a '\' to continue an entry on the next line.

Comments may also be included. A '**#**' makes a comment of all further text until the next NEWLINE not immediately preceded by a '\'**.**

Leading and trailing blanks are allowed in any of the fields.

The **device_allocate** file must be created by the system administrator before device allocation is enabled.

The **device_allocate** file is owned by root, with a group of sys, and a mode of 0644.

EXAMPLES

Declare that physical device st0 is a type **st**. **st** is allocatable, and the script used to clean the device after running **deallocate**(1M) is named **/etc/security/lib/st_clean**.

```
        # scsi tape
st0;\
        st;\
        reserved;\
        reserved;\
        alloc;\
        /etc/security/lib/st_clean;\
```

Declare that physical device fd0 is of type **fd**. **fd** is allocatable, and the script used to clean the device after running **deallocate**(1M) is named **/etc/security/lib/fd_clean**.

```
        # floppy drive
fd0;\
        fd;\
        reserved;\
        reserved;\
        alloc;\
        /etc/security/lib/fd_clean;\
```

Note that making a device allocatable means that you need to allocate and deallocate them to use them (with **allocate**(1M) and **deallocate**(1M)). If a device is allocatable, there will be an asterisk (*) in the *alloc* field, and one can use the device without allocating and deallocating it.

FILES

/etc/security/device_allocate Contains list of allocatable devices

SEE ALSO

allocate(1M), **bsmconv**(1M), **deallocate**(1M), **list_devices**(1M)

NAME

device_maps – device_maps file

SYNOPSIS

/etc/security/device_maps

AVAILABILITY

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

DESCRIPTION

The **device_maps** file contains access control information about each physical device. Each device is represented by a one line entry of the form:

device-name **:** *device-type* **:** *device-list* **:**

where

device-name — This is an arbitrary ASCII string naming the physical device. This field contains no embedded white space or non-printable characters.

device-type — This is an arbitrary ASCII string naming the generic device type. This field identifies and groups together devices of like type. This field contains no embedded white space or non-printable characters.

device-list — This is a list of the device special files associated with the physical device. This field contains valid device special file path names separated by white space.

The **device_maps** file is an ASCII file that resides in the **/etc/security** directory.

Lines in **device_maps** can end with a '\' to continue an entry on the next line.

Comments may also be included. A '**#**' makes a comment of all further text until the next NEWLINE not immediately preceded by a '\'**.**

Leading and trailing blanks are allowed in any of the fields.

The **device_maps** file must be created by the system administrator before device allocation is enabled.

This file is owned by root, with a group of **sys**, and a mode of 0644.

EXAMPLES

```
        # scsi tape
st1:\
        rmt:\
        /dev/rst21 /dev/nrst21 /dev/rst5 /dev/nrst5 /dev/rst13 \
        /dev/nrst13 /dev/rst29 /dev/nrst29 /dev/rmt/1l /dev/rmt/1m \
        /dev/rmt/1 /dev/rmt/1h /dev/rmt/1u /dev/rmt/1ln /dev/rmt/1mn \
        /dev/rmt/1n /dev/rmt/1hn /dev/rmt/1un /dev/rmt/1b /dev/rmt/1bn:\
```

FILES **/etc/security/device_maps**

SEE ALSO **allocate**(1M), **bsmconv**(1M), **deallocate**(1M), **dminfo**(1M), **list_devices**(1M)

NAME

dfstab – file containing commands for sharing resources across a network

DESCRIPTION

dfstab resides in directory **/etc/dfs** and contains commands for sharing resources across a network. **dfstab** gives a system administrator a uniform method of controlling the automatic sharing of local resources.

Each line of the **dfstab** file consists of a **share**(1M) command. The **dfstab** file can be read by the shell to share all resources. System administrators can also prepare their own shell scripts to execute particular lines from **dfstab**.

The contents of **dfstab** are executed automatically when the system enters run-level 3.

SEE ALSO

share(1M), **shareall**(1M)

NAME

dir_ufs, dir – format of ufs directories

SYNOPSIS

```
#include <sys/param.h>
#include <sys/types.h>
#include <sys/fs/ufs_fsdir.h>
```

DESCRIPTION

A directory consists of some number of blocks of **DIRBLKSIZ** bytes, where **DIRBLKSIZ** is chosen such that it can be transferred to disk in a single atomic operation (for example, 512 bytes on most machines).

Each **DIRBLKSIZ**-byte block contains some number of directory entry structures, which are of variable length. Each directory entry has a **struct direct** at the front of it, containing its inode number, the length of the entry, and the length of the name contained in the entry. These entries are followed by the name padded to a 4 byte boundary with null bytes. All names are guaranteed null-terminated. The maximum length of a name in a directory is **MAXNAMLEN**.

```
#define     DIRBLKSIZ                   DEV_BSIZE
#define     MAXNAMLEN                   256
struct direct {
        u_long  d_ino;                  /* inode number of entry */
        u_short d_reclen;               /* length of this record */
        u_short d_namlen;               /* length of string in d_name */
        char    d_name[MAXNAMLEN + 1];  /* name must be no longer than this */
};
```

SEE ALSO

fs_ufs(4)

NAME

dirent – file system independent directory entry

SYNOPSIS

```
#include <dirent.h>
```

DESCRIPTION

Different file system types may have different directory entries. The **dirent** structure defines a file system independent directory entry, which contains information common to directory entries in different file system types. A set of these structures is returned by the **getdents**(2) system call.

The **dirent** structure is defined:

```
struct  dirent {
        ino_t                   d_ino;
        off_t                   d_off;
        unsigned short          d_reclen;
        char                    d_name[1];
};
```

The **d_ino** is a number which is unique for each file in the file system. The field **d_off** is the byte offset of the next, non-empty directory entry in the *actual file system directory*. The field **d_name** is the beginning of the character array giving the name of the directory entry. This name is null terminated and may have at most **MAXNAMLEN** characters. This results in file system independent directory entries being variable length entities. The value of **d_reclen** is the record length of this entry. This length is defined to be the number of bytes between the current entry and the next one, so that the next structure will be suitably aligned.

SEE ALSO

getdents(2)

NAME

driver.conf – driver configuration files

SYNOPSIS

driver.conf

DESCRIPTION

Driver configuration files pass information about device drivers and their configuration to the system. Most device drivers do not have to have configuration files. Drivers for devices that are self-identifying, such as the SBus devices on many systems, can usually obtain all the information they need from the FCode PROM on the SBus card using the DDI property interfaces. See **ddi_prop_op**(9F) for details.

The system associates a driver with its configuration file by name. For example, a driver in **/usr/kernel/drv** called **wombat** has the driver configuration file **wombat.conf** associated with it. By convention, the driver configuration file lives in the same directory as the driver.

The syntax of a single entry in a driver configuration file takes one of three forms:

name="*node name*" **parent**="*parent name*" [*property-name=value* ...]**;**

In this form, the parent name is a simple nexus driver name. Alternatively, the parent can be specified by the type of interface it presents to its children.

name="*node name*" **class**="*class name*" [*property-name=value* ...]**;**

For example, the driver for the SCSI host adapter may have different names on different platforms, but the target drivers can use class **scsi** to insulate themselves from these differences.

Entries of either form above correspond to a device information (devinfo) node in the kernel device tree. Each node has a *name* which is usually the name of the driver, and a *parent* name which is the name of the parent devinfo node it will be connected to. Any number of name-value pairs may be specified to create properties on the prototype devinfo node. These properties can be sized and retrieved using the DDI property interfaces (for example, **ddi_getproplen**(9F) and **ddi_getprop**(9F)). The prototype devinfo node specification must be terminated with a semicolon (**;**).

The third form of an entry is simply a list of properties.

[*property-name=value* ...]**;**

A property created in this way is treated as global to the driver. It can be overridden by a property with the same name on a particular devinfo node, either by creating one explicitly on the prototype node in the driver.conf file or by the driver.

Items are separated by any number of newlines, SPACE or TAB characters.

The configuration file may contain several entries to specify different device configurations and parent nodes. The system may call the driver for each possible prototype devinfo node, and it is generally the responsibility of the drivers **probe**(9E) routine to determine if the hardware described by the prototype devinfo node is really present.

Property names should obey the same naming convention as Open Boot PROM properties, in particular they should not contain at-sign (**@**), or slash (**/**) characters. Property values can be decimal integers or strings delimited by double quotes ("). Hexadecimal

integers can be constructed by prefixing the digits with **0x**.

A comma separated list of integers can be used to construct properties whose value is an integer array. The value of such properties can be retrieved inside the driver using **ddi_getlongprop**(9F) or one of the related property interfaces.

Comments are specified by placing a **#** character at the beginning of the comment string, the comment string extends for the rest of the line.

EXAMPLES

Here is a configuration file called **ACME,simple.conf** for a VMEbus frame buffer called **ACME,simple**.

```
#
# Copyright (c) 1993, by ACME Fictitious Devices, Inc.
#
#ident  "@(#)ACME,simple.conf        1.3     93/09/09"

name="ACME,simple" class="vme"
     reg=0x7d,0x400000,0x110600;
```

This example creates a prototype devinfo node called **ACME,simple** under all parent nodes of class **vme**. It specifies a property called **reg** that consists of an array of three integers. The **reg** property is interpreted by the parent node, see **vme**(4) for further details.

Here is a configuration file called **ACME,example.conf** for a pseudo device driver called **ACME,example**.

```
#
# Copyright (c) 1993, ACME Fictitious Devices, Inc.
#
#ident  "@(#)ACME,example.conf       1.2     93/09/09"

name="ACME,example" parent="pseudo" instance=0
   debug-level=1;

name="ACME,example" parent="pseudo" instance=1;

whizzy-mode="on";
debug-level=3;
```

This example creates two devinfo nodes called **ACME,example** which will attach below the **pseudo** node in the kernel device tree. The **instance** property is only interpreted by the **pseudo** node, see **pseudo**(4) for further details. A property called **debug-level** will be created on the first devinfo node which will have the value 1. The **example** driver will be able to fetch the value of this property using **ddi_getprop**(9F).

Two global driver properties are created, **whizzy-mode** (which will have the string value "on") and **debug-level** (which will have the value 3). If the driver looks up the property **whizzy-mode** on either node, it will retrieve the value of the global **whizzy-mode** property ("on"). If the driver looks up the **debug-level** property on the first node, it will

retrieve the value of the **debug-level** property on that node (1). Looking up the same property on the second node will retrieve the value of the global **debug-level** property (3).

SEE ALSO

pseudo(4), **sbus**(4), **scsi**(4), **vme**(4), **ddi_getprop**(9F), **ddi_getproplen**(9F), **ddi_getlongprop**(9F), **ddi_prop_op**(9F)

Writing Device Drivers

WARNINGS

To avoid namespace collisions between multiple driver vendors, it is strongly recommended that the *name* property of the driver should begin with a vendor-unique string. A reasonably compact and unique choice is the vendor over-the-counter stock symbol.

NAME

environ, .environ, .pref, .variables – user-preference variables files for AT&T FACE

DESCRIPTION

The **.environ**, **.pref**, and **.variables** files contain variables that indicate user preferences for a variety of operations. The **.environ** and **.variables** files are located under the user's **$HOME/pref** directory. The **.pref** files are found under **$HOME/FILECABINET**, **$HOME/WASTEBASKET**, and any directory where preferences were set via the **organize** command. Names and descriptions for each variable are presented below. Variables are listed one per line and are of the form *variable*=*value*.

Variables found in **.environ** include:

LOGINWIN[**1-4**]
: Windows that are opened when FACE is initialized

SORTMODE
: Sort mode for file folder listings. Values include the following hexadecimal digits:

  1 sorted alphabetically by name

  2 files most recently modified first

  800 sorted alphabetically by object type

  The values above may be listed in reverse order by "ORing" the following value:

  1000 list objects in reverse order. For example, a value of **1002** will produce a folder listing with files LEAST recently modified displayed first. A value of **1001** would produce a "reverse" alphabetical by name listing of the folder

DISPLAYMODE
: Display mode for file folders. Values include the following hexadecimal digits:

  0 file names only

  4 file names and brief description

  8 file names, description, plus additional information

WASTEPROMPT
: Prompt before emptying wastebasket (yes/no)?

WASTEDAYS
: Number of days before emptying wastebasket

PRINCMD[**1-3**]
: Print command defined to print files.

UMASK
: Holds default permissions that files will be created with.

Variables found in **.pref** are the following:

SORTMODE
: which has the same values as the **SORTMODE** variable described in **.environ** above.

DISPMODE
which has the same values as the **DISPLAYMODE** variable described in **.environ** above.

Variables found in **.variables** include:

EDITOR Default editor

PS1 shell prompt

FILES

$HOME/pref/.environ
$HOME/pref/.variables
$HOME/FILECABINET/.pref
$HOME/WASTEBASKET/.pref

NAME

ethers – Ethernet address to hostname database or domain

DESCRIPTION

The **ethers** file is a local source of information about the (48 bit) Ethernet addresses of hosts on the Internet. The **ethers** file can be used in conjunction with or instead of other ethers sources, including the NIS maps **ethers.byname** and **ethers.byaddr** and the NIS+ table **ethers**. Programs use the **ethers**(3N) routines to access this information.

The **ethers** file has one line for each host on an Ethernet. The line has the following format:

Ethernet-address official-host-name

Items are separated by any number of SPACE and/or TAB characters. A '**#**' indicates the beginning of a comment extending to the end of line.

The standard form for Ethernet addresses is "*x*:*x*:*x*:*x*:*x*:*x*" where *x* is a hexadecimal number between 0 and ff, representing one byte. The address bytes are always in network order. Host names may contain any printable character other than SPACE, TAB, NEWLINE, or comment character.

FILES

/etc/ethers

SEE ALSO

ethers(3N), **hosts**(4), **nsswitch.conf**(4)

NAME

fd – file descriptor files

DESCRIPTION

These files, conventionally called **/dev/fd/0**, **/dev/fd/1**, **/dev/fd/2**, and so on, refer to files accessible through file descriptors. If file descriptor *n* is open, these two system calls have the same effect:

fd = open("/dev/fd/*n***",mode);**
fd = dup(*n***);**

On these files **creat**(2) is equivalent to **open**, and **mode** is ignored. As with **dup**, subsequent reads or writes on **fd** fail unless the original file descriptor allows the operations.

For convenience in referring to standard input, standard output, and standard error, an additional set of names is provided: **/dev/stdin** is a synonym for **/dev/fd/0**, **/dev/stdout** for **/dev/fd/1**, and **/dev/stderr** for **/dev/fd/2**.

SEE ALSO

creat(2), **dup**(2), **open**(2)

DIAGNOSTICS

open(2) returns –**1** and **EBADF** if the associated file descriptor is not open.

NAME

filehdr – file header for common object files

SYNOPSIS

```
#include <filehdr.h>
```

DESCRIPTION

Every common object file begins with a 20-byte header. The following C **struct** declaration is used:

```
struct  filehdr
{
  unsigned short  f_magic ;   /* magic number */
  unsigned short  f_nscns ;   /* number of sections */
  long            f_timdat ;  /* time & date stamp */
  long            f_symptr ;  /* file ptr to symtab */
  long            f_nsyms ;   /* number of symtab entries */
  unsigned short  f_opthdr ;  /* sizeof(opt and header) */
  unsigned short  f_flags ;   /* flags */
};
```

f_symptr is the byte offset into the file at which the symbol table can be found. Its value can be used as the offset in **fseek**(3S) to position an I/O stream to the symbol table. The UNIX system optional header is 28 bytes. The valid magic numbers are given below:

```
#define  I386MAGIC      0514    /* i386 Computer */
#define  WE32MAGIC      0560    /* 3B2, 3B5, and 3B15 computers */
#define  N3BMAGIC       0550    /* 3B20 computer */
#define  NTVMAGIC       0551    /* 3B20 computer */

#define  VAXWRMAGIC     0570    /* VAX writable text segments */
#define  VAXROMAGIC     0575    /* VAX read only sharable
                                   text segments */
```

The value in **f_timdat** is obtained from the **time**(2) system call. Flag bits currently defined are:

```
#define  F_RELFLG       0000001    /* relocation entries stripped */
#define  F_EXEC         0000002    /* file is executable */
#define  F_LNNO         0000004    /* line numbers stripped */
#define  F_LSYMS        0000010    /* local symbols stripped */
#define  F_AR16WR       0000200    /* 16-bit DEC host */
#define  F_AR32WR       0000400    /* 32-bit DEC host */
#define  F_AR32W        0001000    /* non-DEC host */
#define  F_BM32ID       0160000    /* WE32000 family ID field */
```

```
#define  F_BM32B      0020000    /* file contains WE 32100 code */
#define  F_BM32MAU    0040000    /* file reqs MAU to execute */
#define  F_BM32RST    0010000    /* this object file contains restore
                                    work around [3B5/3B2 only] */
```

SEE ALSO **time**(2), **fseek**(3S), **a.out**(4)

NAME

format.dat – disk drive configuration for the format command.

DESCRIPTION

format.dat enables you to use your specific disk drives with **format**(1M). On Solaris 2.3 and later systems, **format** will automatically configure and label SCSI drives, so that they need not be defined in **format.dat**. Three things can be defined in the data file:

- search paths
- disk types
- partition tables.

Syntax

The following syntax rules apply to the data file:

- The pound **#** sign is the comment character. Any text on a line after a pound sign is not interpreted by **format**.
- Each definition in the **format.dat** file appears on a signle logical line. If the definition is more than one line long, all but the last line of the definition must end with a backslash (\).
- A definition consists of a series of assignments that have an identifier on the left side and one or more values on the right side. The assignment operator is the equal sign (=). Assignments within a definition must be separated by a colon (:).
- White space is ignored by **format**(1M). If you want an assigned value to contain white space, enclose the entire value in double quotes ("). This will cause the white space within quotes to be preserved as part of the assignment value.
- Some assignments can have multiple values on the right hand side. Separate values by a comma (,).

Keywords

The data file contains disk definitions that are read in by **format**(1M) when it starts up. Each definition starts with one of the following keywords: **search_path**, **disk_type**, and **partition**.

search_path 4.x: Tells **format** which disks it should search for when it starts up. The list in the default data file contains all the disks in the GENERIC configuration file. If your system has disks that are not in the GENERIC configuration file, add them to the **search_path** definition in your data file. The data file can contain only one **search_path** definition. However, this single definition lets you specify all the disks you have in your system.

5.x: By default, **format**(1M) understands all the logical devices that are of the form **/dev/rdsk/c***n***t***n***d***n***s***n*; hence **search_path** is not normally defined on a 5.x system.

disk_type Defines the controller and disk model. Each **disk_type** definition contains information concerning the physical geometry of the disk. The default data file contains definitions for the controllers and disks that the Solaris operating system supports. You need to add a new **disk_type** only if you have an unsupported disk. You can add as many **disk_type**

definitions to the data file as you want.

The following controller types are supported by **format**(1M):

XY450	Xylogics 450 controller (SMD)
XD7053	Xylogics 7053 controller (SMD)
MD21	SCSI, but using ESDI devices (aka shoebox)
SCSI	True SCSI (CCS or SCSI-2)
ISP-80	IPI panther controller

Note: The **disk_type** and **partition** definition entries must have **"ctlr = MD21"** for scsi disk devices for 4.1.1 release. But for 4.1.2, 4.1.3 and 5.x releases, the entries should say **"ctlr = SCSI"**.

The keyword itself is assigned the name of the disk type. This name appears in the disk's label and is used to identify the disk type whenever **format**(1M) is run. Enclose the name in double quotes to preserve any white space in the name.

Below are lists of identifiers for supported controllers. Note that an asterisk ('*') indicates the identifier is mandatory for that controller -- it is not part of the keyword name.

The following identifiers are assigned values in all **disk_type** definitions:

acyl*	alternate cylinders
asect	alternate sectors per track
atrks	alternate tracks
fmt_time	formatting time per cylinder
ncyl*	number of logical cylinders
nhead*	number of logical heads
nsect*	number of logical sectors per track
pcyl*	number of physical cylinders
phead	number of physical heads
psect	number of physical sectors per track
rpm*	drive RPM

These identifiers are for SCSI and MD-21 Controllers

read_retries	page 1 byte 3 (read retries)
write_retries	page 1 byte 8 (write retries)
cyl_skew	page 3 bytes 18-19 (cylinder skew)
trk_skew	page 3 bytes 16-17 (track skew)
trks_zone	page 3 bytes 2-3 (tracks per zone)
cache	page 38 byte 2 (cache parameter)
prefetch	page 38 byte 3 (prefetch parameter)
max_prefetch	page 38 byte 4 (minimum prefetch)
min_prefetch	page 38 byte 6 (maximum prefetch)

Note: The Page 38 values are device-specific. Refer the user to the particular disk's manual for these values.

For SCSI disks, the following geometry specifiers may cause a mode select on the byte(s) indicated:

asect	page 3 bytes 4-5 (alternate sectors per zone)
atrks	page 3 bytes 8-9 (alt. tracks per logical unit)
phead	page 4 byte 5 (number of heads)
psect	page 3 bytes 10-11 (sectors per track)

And these identifiers are for SMD Controllers Only

bps*	bytes per sector (SMD)
bpt*	bytes per track (SMD)

Note: under SunOS 5.x, bpt is only required for SMD disks. Under SunOS 4.x, bpt was required for all disk types, even though it was only used for SMD disks.

And this identifier is for XY450 SMD Controllers Only

drive_type*	drive type (SMD) (just call this "xy450 drive type")

partition Defines a partition table for a specific disk type. The partition table contains the partitioning information, plus a name that lets you refer to it in **format**(1M). The default data file contains default parition definitions for several kinds of disk drives. Add a partition definition if you repartitioned any of the disks on your system. Add as many partition definitions to the data file as you need.

Partition naming conventions differ in SunOS 4.x and in SunOS 5.x.

4.x: the partitions are named as **a**, **b**, **c**, **d**, **e**, **f**, **g**, **h**.

5.x: the partitions are referred to by numbers **0**, **1**, **2**, **3**, **4**, **5**, **6**, **7**.

EXAMPLES

Following is a sample **disk_type** and **partition** definition in **format.dat** file for SUN0535 disk device.

```
disk_type = "SUN0535" \
        : ctlr = SCSI : fmt_time = 4 \
        : ncyl = 1866 : acyl = 2 : pcyl = 2500 : nhead = 7 : nsect = 80 \
        : rpm = 5400

partition = "SUN0535" \
        : disk = "SUN0535" : ctlr = SCSI \
        : 0 = 0, 64400 : 1 = 115, 103600 : 2 = 0, 1044960 : 6 = 300, 876960
```

FILES

/etc/format.dat	default data file if **format –x** is not specified, nor is there a **format.dat** file in the current directory.

SEE ALSO

format(1M)
Peripherals Administration

NAME

fs_ufs, inode_ufs, inode – format of a ufs file system volume

SYNOPSIS

```
#include <sys/param.h>
#include <sys/types.h>
#include <sys/fs/ufs_fs.h>
#include <sys/fs/ufs_inode.h>
```

DESCRIPTION

Standard ufs file system storage volumes have a common format for certain vital information. Every volume is divided into a certain number of blocks. The block size is a parameter of the file system. Sectors 0 to 15 contain primary and secondary bootstrapping programs.

The actual file system begins at sector 16 with the super-block. The layout of the super-block is defined by the header **<sys/fs/ufs_fs.h>**.

Each disk drive contains some number of file systems. A file system consists of a number of cylinder groups. Each cylinder group has inodes and data.

A file system is described by its super-block, and by the information in the cylinder group blocks. The super-block is critical data and is replicated before each cylinder group block to protect against catastrophic loss. This is done at file system creation time and the critical super-block data does not change, so the copies need not be referenced.

fs_clean indicates the state of the file system. The **FSCLEAN** state indicates an undamaged, cleanly unmounted file system. The **FSACTIVE** state indicates a mounted file system that has been updated. The **FSSTABLE** state indicates an idle mounted file system. It is not necessary to run **fsck** on any unmounted file systems with a state of **FSCLEAN** or **FSSTABLE**. **mount**(2) will return **ENOSPC** if a ufs file system with a state of **FSACTIVE** is being mounted for read-write.

To provide additional safeguard, **fs_clean** could be trusted only if **fs_state** contains a value equal to **FSOKAY - fs_time**, where **FSOKAY** is a constant integer. Otherwise, **fs_clean** is treated as though it contains the value of **FSACTIVE**.

Addresses stored in inodes are capable of addressing fragments of “blocks.” File system blocks of at most, size **MAXBSIZE** can be optionally broken into 2, 4, or 8 pieces, each of which is addressable; these pieces may be **DEV_BSIZE** or some multiple of a **DEV_BSIZE** unit.

Large files consist exclusively of large data blocks. To avoid undue wasted disk space, the last data block of a small file is allocated only as many fragments of a large block as are necessary. The file system format retains only a single pointer to such a fragment, which is a piece of a single large block that has been divided. The size of such a fragment is determinable from information in the inode, using the **blksize(fs, ip, lbn)** macro.

The file system records space availability at the fragment level; aligned fragments are examined to determine block availability.

The root inode is the root of the file system. Inode 0 cannot be used for normal purposes and historically, bad blocks were linked to inode 1. Thus the root inode is 2 (inode 1 is no longer used for this purpose; however numerous dump tapes make this assumption, so we are stuck with it). The *lost+found* directory is given the next available inode when it is initially created by **mkfs**(1M).

fs_minfree gives the minimum acceptable percentage of file system blocks which may be free. If the freelist drops below this level only the super-user may continue to allocate blocks. **fs_minfree** may be set to 0 if no reserve of free blocks is deemed necessary, however severe performance degradations will be observed if the file system is run at greater than 90% full; thus the default value of **fs_minfree** is 10%.

Empirically the best trade-off between block fragmentation and overall disk utilization at a loading of 90% comes with a fragmentation of 8; thus the default fragment size is an eighth of the block size.

fs_optim specifies whether the file system should try to minimize the time spent allocating blocks, or if it should attempt to minimize the space fragmentation on the disk. If the value of **fs_minfree** is less than 10%, then the file system defaults to optimizing for space to avoid running out of full sized blocks. If the value of **fs_minfree** is greater than or equal to 10%, fragmentation is unlikely to be problematical, and the file system defaults to optimizing for time.

Cylinder group related limits: Each cylinder keeps track of the availability of blocks at different rotational positions, so that sequential blocks can be laid out with minimum rotational latency. **fs_nrpos** is the number of rotational positions which are distinguished. With the default **fs_nrpos** of 8, the resolution of the summary information is 2ms for a typical 3600 rpm drive.

fs_rotdelay gives the minimum number of milliseconds to initiate another disk transfer on the same cylinder. It is used in determining the rotationally optimal layout for disk blocks within a file; the default value for **fs_rotdelay** varies from drive to drive (see **tunefs**(1M)).

fs_maxcontig gives the maximum number of blocks, belonging to one file, that will be allocated contiguously before inserting a rotational delay.

Each file system has a statically allocated number of inodes. An inode is allocated for each **NBPI** bytes of disk space. The inode allocation strategy is extremely conservative.

MINBSIZE is the smallest allowable block size. With a **MINBSIZE** of 4096 it is possible to create files of size 2^32 with only two levels of indirection. **MINBSIZE** must be large enough to hold a cylinder group block, thus changes to **(struct cg)** must keep its size within **MINBSIZE**. Note: super-blocks are never more than size **SBSIZE**.

The path name on which the file system is mounted is maintained in **fs_fsmnt**. **MAXMNTLEN** defines the amount of space allocated in the super-block for this name.

The limit on the amount of summary information per file system is defined by **MAXCSBUFS**. It is currently parameterized for a maximum of two million cylinders.

Per cylinder group information is summarized in blocks allocated from the first cylinder group's data blocks. These blocks are read in from **fs_csaddr** (size **fs_cssize**) in addition to the super-block.

Note: **sizeof (struct csum)** must be a power of two in order for the **fs_cs** macro to work.

The inode is the focus of all file activity in the file system. There is a unique inode allocated for each active file, each current directory, each mounted-on file, text file, and the root. An inode is "named" by its device/i-number pair. For further information, see the header **<sys/fs/ufs_inode.h>**.

SEE ALSO **fsck_ufs**(1M), **mkfs_ufs**(1M), **mount**(2)

NAME

fspec – format specification in text files

DESCRIPTION

It is sometimes convenient to maintain text files on the system with non-standard tabs, (tabs that are not set at every eighth column). Such files must generally be converted to a standard format, frequently by replacing all tabs with the appropriate number of spaces, before they can be processed by system commands. A format specification occurring in the first line of a text file specifies how tabs are to be expanded in the remainder of the file.

A format specification consists of a sequence of parameters separated by blanks and surrounded by the brackets **<:** and **:>**. Each parameter consists of a keyletter, possibly followed immediately by a value. The following parameters are recognized:

t*tabs* The **t** parameter specifies the tab settings for the file. The value of **tabs** must be one of the following:

A list of column numbers separated by commas, indicating tabs set at the specified columns

A '–' followed immediately by an integer *n*, indicating tabs at intervals of *n* columns

A '–' followed by the name of a ''canned'' tab specification

Standard tabs are specified by **t–8**, or equivalently, **t1,9,17,25,** etc. The canned tabs that are recognized are defined by the **tabs**(1) command.

s*size* The **s** parameter specifies a maximum line size. The value of *size* must be an integer. Size checking is performed after tabs have been expanded, but before the margin is prepended.

m*margin* The **m** parameter specifies a number of spaces to be prepended to each line. The value of *margin* must be an integer.

d The **d** parameter takes no value. Its presence indicates that the line containing the format specification is to be deleted from the converted file.

e The **e** parameter takes no value. Its presence indicates that the current format is to prevail only until another format specification is encountered in the file.

Default values, which are assumed for parameters not supplied, are **t–8** and **m0**. If the **s** parameter is not specified, no size checking is performed. If the first line of a file does not contain a format specification, the above defaults are assumed for the entire file. The following is an example of a line containing a format specification:

∗ **<:t5,10,15 s72:>** ∗

If a format specification can be disguised as a comment, it is not necessary to code the **d** parameter.

SEE ALSO

ed(1), **newform**(1), **tabs**(1)

NAME

fstypes – file that registers distributed file system packages

DESCRIPTION

fstypes resides in directory **/etc/dfs** and lists distributed file system utilities packages installed on the system. For each installed distributed file system type, there is a line that begins with the file system type name (for example, ‘‘nfs’’), followed by white space and descriptive text.

The file system indicated in the first line of the file is the default file system; when Distributed File System (DFS) Administration commands are entered without the option **–F** *fstypes*, the system takes the file system type from the first line of the **fstypes** file.

The default file system can be changed by editing the **fstypes** file with any supported text editor.

SEE ALSO

dfmounts(1M), **dfshares**(1M), **share**(1M), **shareall**(1M), **unshare**(1M)

NAME

group – group file

DESCRIPTION

The **group** file is a local source of group information. The **group** file can be used in conjunction with other group sources, including the NIS maps **group.byname** and **group.bygid** and the NIS+ table **group**. Programs use the **getgrnam**(3C) routines to access this information.

The **group** file contains a one-line entry for each group recognized by the system, of the form:

groupname:*password*: *gid*:*user-list*

groupname	The name of the group.
gid	The group's unique numerical ID within the system.
user-list	A comma-separated list of users allowed in the group.

If the password field is empty, no password is demanded. During user identification and authentication, the supplementary group access list is initialized sequentially from information in this file. If a user is in more groups than the system is configured for, {NGROUPS_MAX}, a warning will be given and subsequent group specifications will be ignored.

Malformed entries cause routines that read this file to halt, in which case group assignments specified further along are never made. To prevent this from happening, use **grpck**(1B) to check the **/etc/group** database from time to time.

Previous releases used a group entry beginning with a '**+**' (plus sign) or '**–**' (minus sign) to selectively incorporate entries from NIS maps for group. If still required, this is supported by specifying **group:compat** in **nsswitch.conf**(4). The ''compat'' source may not be supported in future releases. The preferred sources are, ''files'' followed by ''nisplus''. This has the effect of incorporating the entire contents of the NIS+ **group** table after the **group** file.

EXAMPLES

Here is a sample **group** file:

```
root::0:root
stooges:q.mJzTnu8icF.:10:larry,moe,curly
```

and the sample group entry from nsswitch.conf:

```
group: files nisplus
```

With these entries, the group **stooges** will have members **larry**, **moe**, and **curly**, and all groups listed in the NIS+ group table are effectively incorporated after the entry for **stooges**.

If the **group** file was:

```
root::0:root
stooges:q.mJzTnu8icF.:10:larry,moe,curly
+:
```

and the group entry from nsswitch.conf:

```
group: compat
```

all the groups listed in the NIS **group.bygid** and **group.byname** maps would be effectively incorporated after the entry for stooges.

SEE ALSO

groups(1), **grpck**(1B), **newgrp**(1M), **getgrnam**(3C), **initgroups**(3C), **nsswitch.conf**(4), **unistd**(4)

NAME

holidays – prime/nonprime table for the accounting system

SYNOPSIS

/etc/acct/holidays

DESCRIPTION

The **/etc/acct/holidays** file describes which hours are considered prime time and which days are holidays. Holidays and weekends are considered non-prime time hours. **/etc/acct/holidays** is used by the accounting system.

All lines beginning with an "*" are comments.

The **/etc/acct/holidays** file consists of two sections. The first non-comment line defines the current year and the start time of prime and non-prime time hours, in the form:

current_year prime_start non_prime_start

The remaining non-comment lines define the holidays in the form:

month/day company_holiday

Of these two fields, only the *month/day* is actually used by the accounting system programs.

The **/etc/acct/holidays** file must be updated each year.

EXAMPLES

The following is an example of the **/etc/acct/holidays** file:

```
* Prime/Nonprime Table for the accounting system
*
* Curr  Prime   Non-Prime
* Year  Start   Start
*
 1991   0830    1800
*
* only the first column (month/day) is significant.
*
* month/day     Company
*               Holiday
*
1/1             New Years Day
5/30            Memorial Day
7/4             Indep. Day
9/5             Labor Day
11/24           Thanksgiving Day
11/25           day after Thanksgiving
12/25           Christmas
12/26           day after Christmas
```

SEE ALSO

acct(1M)

NAME

hosts – host name database

SYNOPSIS

/etc/inet/hosts
/etc/hosts

DESCRIPTION

The **hosts** file is a local database that associates the names of hosts with their Internet Protocol (IP) addresses. The **hosts** file can be used in conjunction with, or instead of, other hosts databases, including the Domain Name System (DNS), the NIS **hosts** map and the NIS+ **hosts** table. Programs use library interfaces to access information in the **hosts** file.

The **hosts** file has one entry for each IP address of each host. If a host has more than one IP address, it will have one entry for each. The format of each line is:

IP-address official-host-name nicknames . . .

Items are separated by any number of SPACE and / or TAB characters. The first item on a line is the host's IP address. The second entry is the host's official name. Subsequent entries on the same line are alternative names for the same machine, or "nicknames." Nicknames are optional. A '**#**' indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines that search the file.

Network addresses are written in the conventional "decimal dot" notation and interpreted using the **inet_addr** routine from the Internet address manipulation library, **inet**(3N). Host names may contain any printable character other than a field delimiter, NEWLINE, or comment character. It is recommended that host names *not* contain '.' (dot).

EXAMPLES

Here is a typical line from the **hosts** file:

192.9.1.20 gaia # John Smith

SEE ALSO

in.named(1M), **gethostbyname**(3N), **inet**(3N), **nsswitch.conf**(4) **resolv.conf**(4),

NOTES

/etc/inet/hosts is the official SVR4 name of the **hosts** file. The symbolic link **/etc/hosts** exists for BSD compatibility.

NAME

hosts.equiv, .rhosts – trusted remote hosts and users

DESCRIPTION

The **/etc/hosts.equiv** and **.rhosts** files provide the ''remote authentication'' database for **rlogin**(1), **rsh**(1), **rcp**(1), and **rcmd**(3N). The files specify remote hosts and users that are considered *trusted*. Trusted users are allowed to access the local system *without supplying a password.* The library routine **ruserok( )** (see **rcmd**(3N)) performs the authentication procedure for programs by using the **/etc/hosts.equiv** and **.rhosts** files. The **/etc/hosts.equiv** file applies to the entire system, while individual users can maintain their own **.rhosts** files in their home directories.

These files *bypass* the standard password-based user authentication mechanism. To maintain system security, care must be taken in creating and maintaining these files.

The remote authentication procedure determines whether a user from a remote host should be allowed to access the local system with the identity of a local user. This procedure first checks the **/etc/hosts.equiv** file and then checks the **.rhosts** file in the home directory of the local user who is requesting access. Entries in these files can be of two forms. *Positive* entries *allow* access, while *negative* entries *deny* access. The authentication succeeds when a matching positive entry is found. The procedure fails when the first matching negative entry is found, or if no matching entries are found in either file. Thus, the order of entries is important; If the files contain positive and negative entries, the entry that appears first will prevail. The **rsh**(1) and **rcp**(1) programs fail if the remote authentication procedure fails. The **rlogin** program falls back to the standard password-based login procedure if the remote authentication fails.

Both files are formatted as a list of one-line entries. Each entry has the form:

hostname [*username*]

Negative entries are differentiated from positive entries by a '–' character preceding either the *hostname* or *username* field.

Positive Entries

If the form:

hostname

is used, then users from the named host are trusted. That is, they may access the system with the same user name as they have on the remote system. This form may be used in both the **/etc/hosts.equiv** and **.rhosts** files.

If the line is in the form:

hostname username

then the named user from the named host can access the system. This form may be used in individual **.rhosts** files to allow remote users to access the system *as a different local user*. If this form is used in the **/etc/hosts.equiv** file, the named remote user will be allowed to access the system as *any* local user.

netgroup(4) can be used in either the *hostname* or *username* fields to match a number of hosts or users in one entry. The form:

+@*netgroup*

allows access from all hosts in the named netgroup. When used in the *username* field, netgroups allow a group of remote users to access the system as a particular local user. The form:

hostname **+@***netgroup*

allows all of the users in the named netgroup from the named host to access the system as the local user. The form:

+@*netgroup1* **+@***netgroup2*

allows the users in *netgroup2* from the hosts in *netgroup1* to access the system as the local user.

The special character '+' can be used in place of either *hostname* or *username* to match any host or user. For example, the entry

+

will allow a user from any remote host to access the system with the same username. The entry

+ *username*

will allow the named user from any remote host to access the system. The entry

hostname **+**

will allow any user from the named host to access the system as the local user.

Negative Entries

Negative entries are preceded by a '/-' sign. The form:

–hostname

will disallow all access from the named host. The form:

*–***@***netgroup*

means that access is explicitly disallowed from all hosts in the named netgroup. The form:

hostname –username

disallows access by the named user only from the named host, while the form:

+ *–***@***netgroup*

will disallow access by all of the users in the named netgroup from all hosts.

FILES

/etc/hosts.equiv
~/.rhosts

SEE ALSO

rcp(1), **rlogin**(1), **rsh**(1), **rcmd**(3N), **hosts**(4), **netgroup**(4), **passwd**(4)

NOTES

Hostnames in **/etc/hosts.equiv** and **.rhosts** files must be the official name of the host, not one of its nicknames.

Root access is handled as a special case. Only the **.rhosts** file is checked when access is being attempted for root. To help maintain system security, the **/etc/hosts.equiv** file is not checked.

As a security feature, the **.rhosts** file must be owned by the user who is attempting access.

Positive entries in **/etc/hosts.equiv** that include a *username* field (either an individual named user, a netgroup, or '**+**' sign) should be used with extreme caution. Because **/etc/hosts.equiv** applies system-wide, these entries allow one, or a group of, remote users to access the system *as any local user*. This can be a security hole.

NAME

inetd.conf – Internet servers database

SYNOPSIS

/etc/inet/inetd.conf
/etc/inetd.conf

DESCRIPTION

The **inetd.conf** file contains the list of servers that **inetd**(1M) invokes when it receives an Internet request over a socket. Each server entry is composed of a single line of the form:

service-name endpoint-type protocol wait-status uid server-program server-arguments

Fields are separated by either SPACE or TAB characters. A '**#**' (number sign) indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines that search this file.

service-name — The name of a valid service listed in the **services** file. For RPC services, the value of the *service-name* field consists of the RPC service name or program number, followed by a '/' (slash) and either a version number or a range of version numbers (for example, **rstatd/2-4**).

endpoint-type — Can be one of:

stream	for a stream socket,
dgram	for a datagram socket,
raw	for a raw socket,
seqpacket	for a sequenced packet socket
tli	for all tli endpoints

protocol — Must be a recognized protocol listed in the file **/etc/inet/protocols**. For RPC services, the field consists of the string **rpc** followed by a '/' (slash) and either a '*' (asterisk), one or more nettypes, one or more netids, or a combination of nettypes and netids. Whatever the value, it is first treated as a nettype. If it is not a valid nettype, then it is treated as a netid. For example, **rpc/***∗ for an RPC service using all the transports supported by the system (the list can be found in the **/etc/netconfig** file), equivalent to saying **rpc/visible rpc/ticots** for an RPC service using the Connection-Oriented Transport Service.

wait-status — **nowait** for all but "single-threaded" datagram servers — servers which do not release the socket until a timeout occurs. These must have the status **wait**. Do not configure **udp** services as **nowait**. This will cause a race condition where the **inetd** program selects on the socket and the server program reads from the socket. Many server programs will be forked and performance will be severly compromised.

uid — The user ID under which the server should run. This allows servers to run with access privileges other than those for root.

server-program — Either the pathname of a server program to be invoked by **inetd** to

perform the requested service, or the value **internal** if **inetd** itself provides the service.

server-arguments — If a server must be invoked with command line arguments, the entire command line (including argument 0) must appear in this field (which consists of all remaining words in the entry). If the server expects **inetd** to pass it the address of its peer (for compatibility with 4.2BSD executable daemons), then the first argument to the command should be specified as '**%A**'. No more than five arguments are allowed in this field.

FILES

/etc/netconfig	network configuration file
/etc/inet/protocols	Internet protocols
/etc/inet/services	Internet network services

SEE ALSO

rlogin(1), **rsh**(1), **in.tftpd**(1M), **inetd**(1M), **services**(4)

NOTES

/etc/inet/inetd.conf is the official SVR4 name of the **inetd.conf** file. The symbolic link **/etc/inetd.conf** exists for BSD compatibility.

NAME

init.d – initialization and termination scripts for changing init states

SYNOPSIS

/etc/init.d

DESCRIPTION

/etc/init.d is a directory containing initialization and termination scripts for changing init states. These scripts are linked when appropriate to files in the **rc?.d** directories, where '**?**' is a single character corresponding to the init state. See **init**(1M) for definitions of the states.

File names in **rc?.d** directories are of the form **[SK]nn<init.d filename>**, where **S** means start this job, **K** means kill this job, and **nn** is the relative sequence number for killing or starting the job. When entering a state (init S,0,2,3,etc.) the **rc[S0-6]** script executes those scripts in **/etc/rc[S0-6].d** that are prefixed with **K** followed by those scripts prefixed with **S**. When executing each script in one of the **/etc/rc[S0-6] directories, the /sbin/rc[S0-6]** script passes a single argument. It passes the argument 'stop' for scripts prefixed with **K** and the argument 'start' for scripts prefixed with **S**. There is no harm in applying the same sequence number to multiple scripts. In this case the order of execution is deterministic but unspecified.

Guidelines for selecting sequence numbers are provided in **README** files located in the directory associated with that target state. For example, **/etc/rc[S0-6].d/README**. Absence of a **README** file indicates that there are currently no established guidelines.

EXAMPLES

When changing to init state 2 (multi-user mode, network resources not exported), **/sbin/rc2** is initiated by the init process. The following steps are performed by **/sbin/rc2**.

1. In the directory **/etc/rc2.d** are files used to stop processes that should not be running in state 2. The filenames are prefixed with **K**. Each **K** file in the directory is executed (by **/sbin/rc2**) in alpha-numeric order when the system enters init state 2. See example below.
2. Also in the **rc2.d** directory are files used to start processes that should be running in state 2. As in the Step 1, each **S** file is executed.

   Assume the file **/etc/netdaemon** is a script that will initiate networking daemons when given the argument argument 'stop'. It is linked to **/etc/rc2.d/S68netdaemon**, and to **/etc/rc0.d/K67netdaemon**. The file is executed by **/etc/rc2.d/S68netdaemon start** when init state 2 is entered and by **/etc/rc0.d/S67netdaemon stop** when shutting the system down.

SEE ALSO

init(1M)

NOTES

/sbin/rc2 has references to the obsolescent **rc.d** directory. These references are for compatibility with old **INSTALL** scripts. New **INSTALL** scripts should use the **init.d** directory for related executables. The same is true for the **shutdown.d** directory.

NAME

inittab – script for init

DESCRIPTION

The file **/etc/inittab** controls process dispatching by **init**. The processes most typically dispatched by **init** are daemons.

The **inittab** file is composed of entries that are position dependent and have the following format:

id:*rstate*:*action*:*process*

Each entry is delimited by a newline; however, a backslash (\ ) preceding a newline indicates a continuation of the entry. Up to 512 characters for each entry are permitted. Comments may be inserted in the *process* field using the convention for comments described in **sh**(1). There are no limits (other than maximum entry size) imposed on the number of entries in the **inittab** file. The entry fields are:

id One or two characters used to uniquely identify an entry.

rstate Define the run level in which this entry is to be processed. Run-levels effectively correspond to a configuration of processes in the system. That is, each process spawned by **init** is assigned a run level(s) in which it is allowed to exist. The run levels are represented by a number ranging from **0** through **6**. For example, if the system is in run level **1**, only those entries having a **1** in the *rstate* field are processed.

When **init** is requested to change run levels, all processes that do not have an entry in the *rstate* field for the target run level are sent the warning signal **SIGTERM** and allowed a 5-second grace period before being forcibly terminated by the kill signal **SIGKILL**. The *rstate* field can define multiple run levels for a process by selecting more than one run level in any combination from **0** through **6**. If no run level is specified, then the process is assumed to be valid at all run levels **0** through **6**.

There are three other values, **a**, **b** and **c**, which can appear in the *rstate* field, even though they are not true run levels. Entries which have these characters in the *rstate* field are processed only when an **init** or **telinit** process requests them to be run (regardless of the current run level of the system). See **init**(1M). These differ from run levels in that **init** can never enter run level **a**, **b** or **c**. Also, a request for the execution of any of these processes does not change the current run level. Furthermore, a process started by an **a**, **b** or **c** command is not killed when **init** changes levels. They are killed only if their line in **inittab** is marked **off** in the *action* field, their line is deleted entirely from **inittab**, or **init** goes into single-user state.

action Key words in this field tell **init** how to treat the process specified in the *process* field. The actions recognized by **init** are as follows:

respawn If the process does not exist, then start the process; do not wait for its termination (continue scanning the **inittab** file), and when the process dies, restart the process. If the process currently exists, do nothing and continue scanning the **inittab** file.

wait When **init** enters the run level that matches the entry's *rstate*, start the process and wait for its termination. All subsequent reads of the **inittab** file while **init** is in the same run level cause **init** to ignore this entry.

once When **init** enters a run level that matches the entry's *rstate*, start the process, do not wait for its termination. When it dies, do not restart the process. If **init** enters a new run level and the process is still running from a previous run level change, the program is not restarted.

boot The entry is to be processed only at **init**'s boot-time read of the **inittab** file. **init** is to start the process and not wait for its termination; when it dies, it does not restart the process. In order for this instruction to be meaningful, the *rstate* should be the default or it must match **init**'s run level at boot time. This action is useful for an initialization function following a hardware reboot of the system.

bootwait The entry is to be processed the first time **init** goes from single-user to multi-user state after the system is booted. (If **initdefault** is set to **2**, the process runs right after the boot.) **init** starts the process, waits for its termination and, when it dies, does not restart the process.

powerfail Execute the process associated with this entry only when **init** receives a power fail signal, **SIGPWR** (see **signal**(3C)).

powerwait Execute the process associated with this entry only when **init** receives a power fail signal, **SIGPWR**, and wait until it terminates before continuing any processing of **inittab**.

off If the process associated with this entry is currently running, send the warning signal **SIGTERM** and wait 5 seconds before forcibly terminating the process with the kill signal **SIGKILL**. If the process is nonexistent, ignore the entry.

ondemand This instruction is really a synonym for the **respawn** action. It is functionally identical to **respawn** but is given a different keyword in order to divorce its association with run levels. This instruction is used only with the **a**, **b** or **c** values described in the *rstate* field.

initdefault An entry with this action is scanned only when **init** is initially invoked. **init** uses this entry to determine which run level to enter initially. It does this by taking the highest run level specified in the *rstate* field and using that as its initial state. If the *rstate* field is empty, this is interpreted as **0123456** and **init** will enter run level **6**. This will cause the system to loop (it will go to firmware and reboot continuously). Additionally, if **init** does not find an **initdefault** entry in **inittab**, it requests an initial run level from the user at reboot time.

sysinit Entries of this type are executed before **init** tries to access the console (that is, before the **Console Login:** prompt). It is expected that this entry will be used only to initialize devices that **init** might try to ask the run level question. These entries are executed and **init** waits for their completion before continuing.

process Specify a command to be executed. The entire **process** field is prefixed with **exec** and passed to a forked **sh** as **sh –c 'exec** *command*'. For this reason, any legal **sh** syntax can appear in the *process* field.

SEE ALSO **sh**(1), **who**(1), **init**(1M), **ttymon**(1M), **exec**(2), **open**(2), **signal**(3C)

NAME

issue – issue identification file

DESCRIPTION

The file **/etc/issue** contains the issue or project identification to be printed as a login prompt. **issue** is an ASCII file that is read by program **getty** and then written to any terminal spawned or respawned from the *lines* file.

FILES

/etc/issue

SEE ALSO

login(1)

NAME

keytables – keyboard table descriptions for loadkeys and dumpkeys

AVAILABILITY

SPARC

DESCRIPTION

These files are used by **loadkeys**(1) to modify the translation tables used by the keyboard streams module and generated by (see **loadkeys**(1)) from those translation tables.

Any line in the file beginning with **#** is a comment, and is ignored. **#** is treated specially only at the beginning of a line.

Other lines specify the values to load into the tables for a particular keystation. The format is either:

key *number list_of_entries*

or

swap *number1* **with** *number2*

or

key *number1* **same as** *number2*

or a blank line, which is ignored.

key *number list_of_entries*

sets the entries for keystation *number* from the list given. An entry in that list is of the form

tablename code

where *tablename* is the name of a particular translation table, or **all**. The translation tables are:

base	entry when no shifts are active
shift	entry when "Shift" key is down
caps	entry when "Caps Lock" is in effect
ctrl	entry when "Control" is down
altg	entry when "Alt Graph" is down
numl	entry when "Num Lock" is in effect
up	entry when a key goes up

All tables other than **up** refer to the action generated when a key goes down. Entries in the **up** table are used only for shift keys, since the shift in question goes away when the key goes up, except for keys such as "Caps Lock" or "Num Lock"; the keyboard streams module makes the key look as if it were a latching key.

A table name of **all** indicates that the entry for all tables should be set to the specified value, with the following exception: for entries with a value other than **hole**, the entry for the **numl** table should be set to **nonl**, and the entry for the **up** table should be set to **nop**.

The *code* specifies the effect of the key in question when the specified shift key is down. A *code* consists of either:

- A character, which indicates that the key should generate the given character. The character can either be a single character, a single character preceded by ^ which refers to a "control character" (for instance, **^c** is control-C), or a C-style character constant enclosed in single quote characters (**'**), which can be expressed with C-style escape sequences such as \r for RETURN or \000 for the null character. Note that the single character may be any character in an 8-bit character set, such as ISO 8859/1.
- A string, consisting of a list of characters enclosed in double quote characters ("). Note that the use of the double quote character means that a *code* of double quote must be enclosed in single quotes.
- One of the following expressions:

shiftkeys+leftshift
the key is to be the left-hand "Shift" key

shiftkeys+rightshift
the key is to be the right-hand "Shift" key

shiftkeys+leftctrl
the key is to be the left-hand "Control" key

shiftkeys+rightctrl
the key is to be the right-hand "Control" key

shiftkeys+alt
the key is to be the "Alt" shift key

shiftkeys+altgraph
the key is to be the "Alt Graph" shift key

shiftkeys+capslock
the key is to be the "Caps Lock" key

shiftkeys+shiftlock
the key is to be the "Shift Lock" key

shiftkeys+numlock
the key is to be the "Num Lock" key

buckybits+systembit
the key is to be the "Stop" key in SunView; this is normally the L1 key, or the SETUP key on the VT100 keyboard

buckybits+metabit
the key is to be the "meta" key. That is, the "Left" or "Right" key on a Sun-2 or Sun-3 keyboard or the "diamond" key on a Sun-4 keyboard

compose
the key is to be the "Compose" key

ctrlq on the "VT100" keyboard, the key is to transmit the control-Q character (this would be the entry for the "Q" key in the **ctrl** table)

ctrls on the "VT100" keyboard, the key is to transmit the control-S character (this would be the entry for the "S" key in the **ctrl** table)

noscroll
on the "VT100" keyboard, the key is to be the "No Scroll" key

string+uparrow
the key is to be the "up arrow" key

string+downarrow
the key is to be the "down arrow" key

string+leftarrow
the key is to be the "left arrow" key

string+rightarrow
the key is to be the "right arrow" key

string+homearrow
the key is to be the "home" key

fa_acute
the key is to be the acute accent "floating accent" key

fa_cedilla
the key is to be the cedilla "floating accent" key

fa_cflex
the key is to be the circumflex "floating accent" key

fa_grave
the key is to be the grave accent "floating accent" key

fa_tilde
the key is to be the tilde "floating accent" key

fa_umlaut
the key is to be the umlaut "floating accent" key

nonl this is used only in the Num Lock table; the key is not to be affected by the state of Num Lock

pad0 the key is to be the "0" key on the numeric keypad

pad1 the key is to be the "1" key on the numeric keypad

pad2 the key is to be the "2" key on the numeric keypad

pad3 the key is to be the "3" key on the numeric keypad

pad4 the key is to be the "4" key on the numeric keypad

pad5 the key is to be the "5" key on the numeric keypad

pad6 the key is to be the "6" key on the numeric keypad

pad7 the key is to be the "7" key on the numeric keypad

pad8 the key is to be the "8" key on the numeric keypad

pad9 the key is to be the "9" key on the numeric keypad

paddot the key is to be the "." key on the numeric keypad

padenter
the key is to be the "Enter" key on the numeric keypad

padplus
the key is to be the "+" key on the numeric keypad

padminus
the key is to be the "-" key on the numeric keypad

padstar
the key is to be the "*" key on the numeric keypad

padslash
the key is to be the "/" key on the numeric keypad

padequal
the key is to be the "=" key on the numeric keypad

padsep the key is to be the "," (separator) key on the numeric keypad

lf(*n*) the key is to be the left-hand function key *n*

rf(*n*) the key is to be the right-hand function key *n*

tf(*n*) the key is to be the top function key *n*

bf(*n*) the key is to be the "bottom" function key *n*

nop the key is to do nothing

error this code indicates an internal error; to be used only for keystation 126, and must be used there

idle this code indicates that the keyboard is idle (that is, has no keys down); to be used only for all entries other than the **numl** and **up** table entries for keystation 127, and must be used there

oops this key exists, but its action is not defined; it has the same effect as **nop**

reset this code indicates that the keyboard has just been reset; to be used only for the **up** table entry for keystation 127, and must be used there.

swap *number1* **with** *number2*
exchanges the entries for keystations *number1* and *number2*.

key *number1* **same as** *number2*
sets the entries for keystation *number1* to be the same as those for keystation *number2*. If the file does not specify entries for keystation *number2*, the entries currently in the translation table are used; if the file does specify entries for keystation *number2*, those entries are used.

EXAMPLES

The following entry sets keystation 15 to be a “hole” (that is, an entry indicating that there is no keystation 15); sets keystation 30 to do nothing when Alt Graph is down, generate "!" when Shift is down, and generate "1" under all other circumstances; and sets keystation 76 to be the left-hand Control key.

```
key 15    all hole
key 30    base 1 shift ! caps 1 ctrl 1 altg nop
key 76    all shiftkeys+leftctrl up shiftkeys+leftctrl
```

The following entry exchanges the Delete and Back Space keys on the Type 4 keyboard:

```
swap 43 with 66
```

Keystation 43 is normally the Back Space key, and keystation 66 is normally the Delete key.

The following entry disables the Caps Lock key on the Type 3 and U.S. Type 4 keyboards:

```
key 119 all nop
```

The following specifies the standard translation tables for the U.S. Type 4 keyboard:

```
key 0     all hole
key 1     all buckybits+systembit up buckybits+systembit
key 2     all hole
key 3     all lf(2)
key 4     all hole
key 5     all tf(1)
key 6     all tf(2)
key 7     all tf(10)
key 8     all tf(3)
key 9     all tf(11)
key 10    all tf(4)
key 11    all tf(12)
key 12    all tf(5)
key 13    all shiftkeys+altgraph up shiftkeys+altgraph
key 14    all tf(6)
key 15    all hole
key 16    all tf(7)
key 17    all tf(8)
key 18    all tf(9)
key 19    all shiftkeys+alt up shiftkeys+alt
key 20    all hole
key 21    all rf(1)
key 22    all rf(2)
key 23    all rf(3)
key 24    all hole
key 25    all lf(3)
key 26    all lf(4)
key 27    all hole
key 28    all hole
key 29    all ^[
key 30    base 1 shift ! caps 1 ctrl 1 altg nop
key 31    base 2 shift @ caps 2 ctrl ^@ altg nop
key 32    base 3 shift # caps 3 ctrl 3 altg nop
```

```
key 33   base 4 shift $ caps 4 ctrl 4 altg nop
key 34   base 5 shift % caps 5 ctrl 5 altg nop
key 35   base 6 shift ^ caps 6 ctrl ^^ altg nop
key 36   base 7 shift & caps 7 ctrl 7 altg nop
key 37   base 8 shift * caps 8 ctrl 8 altg nop
key 38   base 9 shift ( caps 9 ctrl 9 altg nop
key 39   base 0 shift ) caps 0 ctrl 0 altg nop
key 40   base - shift _ caps - ctrl ^_ altg nop
key 41   base = shift + caps = ctrl = altg nop
key 42   base ` shift ~ caps ` ctrl ^^ altg nop
key 43   all '\b'
key 44   all hole
key 45   all rf(4) numl padequal
key 46   all rf(5) numl padslash
key 47   all rf(6) numl padstar
key 48   all bf(13)
key 49   all lf(5)
key 50   all bf(10) numl padequal
key 51   all lf(6)
key 52   all hole
key 53   all '\t'
key 54   base q shift Q caps Q ctrl ^Q altg nop
key 55   base w shift W caps W ctrl ^W altg nop
key 56   base e shift E caps E ctrl ^E altg nop
key 57   base r shift R caps R ctrl ^R altg nop
key 58   base t shift T caps T ctrl ^T altg nop
key 59   base y shift Y caps Y ctrl ^Y altg nop
key 60   base u shift U caps U ctrl ^U altg nop
key 61   base i shift I caps I ctrl '\t' altg nop
key 62   base o shift O caps O ctrl ^O altg nop
key 63   base p shift P caps P ctrl ^P altg nop
key 64   base [ shift { caps [ ctrl ^[ altg nop
key 65   base ] shift } caps ] ctrl ^] altg nop
key 66   all '\177'
key 67   all compose
key 68   all rf(7) numl pad7
key 69   all rf(8) numl pad8
key 70   all rf(9) numl pad9
key 71   all bf(15) numl padminus
key 72   all lf(7)
key 73   all lf(8)
key 74   all hole
key 75   all hole
key 76   all shiftkeys+leftctrl up shiftkeys+leftctrl
key 77   base a shift A caps A ctrl ^A altg nop
```

```
key 78   base s shift S caps S ctrl ^S altg nop
key 79   base d shift D caps D ctrl ^D altg nop
key 80   base f shift F caps F ctrl ^F altg nop
key 81   base g shift G caps G ctrl ^G altg nop
key 82   base h shift H caps H ctrl '\b' altg nop
key 83   base j shift J caps J ctrl '\n' altg nop
key 84   base k shift K caps K ctrl '\v' altg nop
key 85   base l shift L caps L ctrl ^L altg nop
key 86   base ; shift : caps ; ctrl ; altg nop
key 87   base '\'' shift '"' caps '\'' ctrl '\'' altg nop
key 88   base '\\' shift | caps '\\' ctrl ^\ altg nop
key 89   all '\r'
key 90   all bf(11) numl padenter
key 91   all rf(10) numl pad4
key 92   all rf(11) numl pad5
key 93   all rf(12) numl pad6
key 94   all bf(8) numl pad0
key 95   all lf(9)
key 96   all hole
key 97   all lf(10)
key 98   all shiftkeys+numlock
key 99   all shiftkeys+leftshift up shiftkeys+leftshift
key 100  base z shift Z caps Z ctrl ^Z altg nop
key 101  base x shift X caps X ctrl ^X altg nop
key 102  base c shift C caps C ctrl ^C altg nop
key 103  base v shift V caps V ctrl ^V altg nop
key 104  base b shift B caps B ctrl ^B altg nop
key 105  base n shift N caps N ctrl ^N altg nop
key 106  base m shift M caps M ctrl '\r' altg nop
key 107  base , shift < caps , ctrl , altg nop
key 108  base . shift > caps . ctrl . altg nop
key 109  base / shift ? caps / ctrl ^_ altg nop
key 110  all shiftkeys+rightshift up shiftkeys+rightshift
key 111  all '\n'
key 112  all rf(13) numl pad1
key 113  all rf(14) numl pad2
key 114  all rf(15) numl pad3
key 115  all hole
key 116  all hole
key 117  all hole
key 118  all lf(16)
key 119  all shiftkeys+capslock
key 120  all buckybits+metabit up buckybits+metabit
key 121  base ' ' shift ' ' caps ' ' ctrl ^@ altg ' '
```

```
key 122  all buckybits+metabit up buckybits+metabit
key 123  all hole
key 124  all hole
key 125  all bf(14) numl padplus
key 126  all error numl error up hole
key 127  all idle numl idle up reset
```

SEE ALSO

loadkeys(1)

NAME

krb.conf – Kerberos configuration file

SYNOPSIS

/etc/krb.conf

DESCRIPTION

krb.conf contains configuration information describing the Kerberos realm and the Kerberos key distribution center (KDC) servers for known realms.

krb.conf contains the name of the local realm in the first line, followed by lines indicating realm / host entries. The first token is a realm name, and the second is the hostname of a host running a KDC for that realm. There can be multiple lines for a given realm; the servers are tried in order until an active one is found. The words *admin server* following the hostname indicate that the host also provides an administrative database server. For example:

ATHENA.MIT.EDU
ATHENA.MIT.EDU kerberos-1.mit.edu admin server
ATHENA.MIT.EDU kerberos-2.mit.edu
LCS.MIT.EDU kerberos.lcs.mit.edu admin server

The Kerberos configuration information can also be supplied using the **krb.conf** NIS map. If **/etc/krb.conf** is not found (or the requested information is not found in it), and the system is running NIS, then the information will be obtained from the NIS map. If neither the file nor the NIS map are found, then the Kerberos library will use the *domainname* (as returned by **domainname**(1M)) as the Kerberos realm, and the host **kerberos** as the location of the KDC. There is no default for the admin server.

Note that every time **krb.conf** is modified, **kerbd**(1M) needs to be restarted.

SEE ALSO

domainname(1M), **kerbd**(1M), **ypmake**(1M), **krb.realms**(4)

BUGS

There is no NIS+ support yet for the **krb.conf** map.

NAME

krb.realms – host to Kerberos realm translation file

SYNOPSIS

/etc/krb.realms

DESCRIPTION

krb.realms provides a translation from a hostname to the Kerberos realm name for the services provided by that host.

Each line of the translation file is in one of the following forms:

host_name kerberos_realm
domain_name kerberos_realm

domain_name should be of the form *.XXX.YYY*, for example, **.LCS.MIT.EDU**.

If a hostname exactly matches the *host_name* field in a line of the first form, the corresponding *kerberos_realm* is used as the realm of the host. If a hostname does not match any *host_name* in the file, but its domain exactly matches the *domain_name* field in a line of the second form, the corresponding *kerberos_realm* is used as the realm of the host.

If no translation entry applies, the host's realm is considered to be the hostname's domain portion converted to upper case.

SEE ALSO

krb_realmofhost(3N)

BUGS

There is no NIS or NIS+ support for this information.

NAME

limits – header for implementation-specific constants

SYNOPSIS

#include <limits.h>

DESCRIPTION

The header **<limits.h>** is a list of minimal magnitude limitations imposed by a specific implementation of the operating system.

ARG_MAX	**1048320**	**/* max length of arguments to exec */**
CHAR_BIT	**8**	**/* max # of bits in a "char" */**
CHAR_MAX	**255**	**/* max value of a "char" */**
CHAR_MIN	**0**	**/* min value of a "char" */**
CHILD_MAX	**25**	**/* max # of processes per user id */**
CLK_TCK	**_sysconf(3)**	**/* clock ticks per second */**
DBL_DIG	**15**	**/* digits of precision of a "double" */**
DBL_MAX	**1.7976931348623157E+308**	**/* max decimal value of a "double"*/**
DBL_MIN	**2.2250738585072014E-308**	**/* min decimal value of a "double"*/**
FCHR_MAX	**1048576**	**/* historical default file size limit in bytes */**
FLT_DIG	**6**	**/* digits of precision of a "float" */**
FLT_MAX	**3.40282347e+38F**	**/* max decimal value of a "float" */**
FLT_MIN	**1.17549435E-38F**	**/* min decimal value of a "float" */**
INT_MAX	**2147483647**	**/* max value of an "int" */**
INT_MIN	**(-2147483647-1)**	**/* min value of an "int" */**
LINK_MAX	**1000**	**/* max # of links to a single file */**
LOGNAME_MAX	**8**	**/* max # of characters in a login name */**
LONG_BIT	**32**	**/* # of bits in a "long" */**
LONG_MAX	**2147483647**	**/* max value of a "long int" */**
LONG_MIN	**(-2147483647-1)**	**/* min value of a "long int" */**
MAX_CANON	**256**	**/* max bytes in a line for canonical processing */**
MAX_INPUT	**512**	**/* max size of a char input buffer */**
MB_LEN_MAX	**5**	**/* max # of bytes in a multibyte character */**
NAME_MAX	**14**	**/* max # of characters in a file name */**
NGROUPS_MAX	**16**	**/* max # of groups for a user */**
NL_ARGMAX	**9**	**/* max value of "digit" in calls to the NLS printf() and scanf() */**
NL_LANGMAX	**14**	**/* max # of bytes in a LANG name */**
NL_MSGMAX	**32767**	**/* max message number */**
NL_NMAX	**1**	**/* max # of bytes in N-to-1 mapping characters */**
NL_SETMAX	**255**	**/* max set number */**
NL_TEXTMAX	**255**	**/* max # of bytes in a message string */**
NZERO	**20**	**/* default process priority */**
OPEN_MAX	**20**	**/* max # of files a process can have open */**
PASS_MAX	**8**	**/* max # of characters in a password */**
PATH_MAX	**1024**	**/* max # of characters in a path name */**
PID_MAX	**30000**	**/* max value for a process ID */**

```
PIPE_BUF        5120            /* max # bytes atomic in write to a pipe */
PIPE_MAX        5120            /* max # bytes written to a pipe
                                in a write */
SCHAR_MAX       127             /* max value of a "signed char" */
SCHAR_MIN       (-128)          /* min value of a "signed char" */
SHRT_MAX        32767           /* max value of a "short int" */
SHRT_MIN        (-32768)        /* min value of a "short int" */
STD_BLK         1024            /* # bytes in a physical I/O block */
SYS_NMLN        257             /* 4.0 size of utsname elements */
                                /* also defined in sys/utsname.h */
SYSPID_MAX      1               /* max pid of system processes */
TMP_MAX         17576           /* max # of unique names generated
                                by tmpnam */
UCHAR_MAX       255             /* max value of an "unsigned char" */
UID_MAX         60000           /* max value for a user or group ID */
UINT_MAX        4294967295      /* max value of an "unsigned int" */
ULONG_MAX       4294967295      /* max value of an "unsigned long int" */
USHRT_MAX       65535           /* max value of an "unsigned short int" */
USI_MAX         4294967295      /* max decimal value of an "unsigned" */
WORD_BIT        32              /* # of bits in a "word" or "int" */
```

The following POSIX definitions are the most restrictive values to be used by a POSIX conformance application. Conforming implementations shall provide values at least this large.

```
_POSIX_ARG_MAX      4096    /* max length of arguments to exec */
_POSIX_CHILD_MAX    6       /* max # of processes per user ID */
_POSIX_LINK_MAX     8       /* max # of links to a single file */
_POSIX_MAX_CANON    255     /* max # of bytes in a line of input */
_POSIX_MAX_INPUT    255     /* max # of bytes in terminal
                            input queue */
_POSIX_NAME_MAX     14      /* # of bytes in a filename */
_POSIX_NGROUPS_MAX  0       /* max # of groups in a process */
_POSIX_OPEN_MAX     16      /* max # of files a process can have open */
_POSIX_PATH_MAX     255     /* max # of characters in a pathname */
_POSIX_PIPE_BUF     512     /* max # of bytes atomic in write
                            to a pipe */
```

NAME

loadfont – format of a font file used as input to the loadfont utility

AVAILABILITY

x86

DESCRIPTION

This section describes the format of files that can be used to change the font used by the console when using the **loadfont**(1) utility with the –**f** option.

The format is compatible with the Binary Distribution Format version 2.1 as developed by Adobe Systems, Inc.; however, certain restrictions apply. Video cards, when used with the Solaris for x86 system in text mode, only accept constant width and constant height fonts in certain sizes.

The **loadfont** utility also requires that there is a description of all 256 characters of the codeset used specified in the fontfile. Certain attributes are not used by **loadfont** but are maintained for compatibility purposes.

File Format

A **loadfont** input file is a plain ASCII file containing only printable characters (octal 40 through 176) and a carriage return at the end of each line.

The information about a particular font should be contained in a single file. The file begins with information on the font in general, followed by the information and bitmaps for the individual characters. The file should contain bitmaps for all 256 characters, and each character should be of the same size.

A font bitmap description file has the following general form, where each item is contained on a separate line of text in the file. Items on a line are separated by spaces:

One or more lines beginning with the word **COMMENT**. These lines can be used to add comments to the file and will be ignored by the **loadfont** program.

The word **STARTFONT** followed by the version number 2.1.

The word **FONT** followed by the full name of the font. The name may continue all the way to the end of the line, and may contain spaces.

The word **SIZE** followed by the point size of the characters, the x resolution, and the y resolution of the font. This line is not used by **loadfont** but it needs to be there for compatibility purposes.

The word **FONTBOUNDINGBOX** followed by the width in x, height in y, and the x and y displacement of the lower left-hand corner from the origin. Again, this line is not used by **loadfont** but it must be there for compatibility purposes.

Optionally, the word **STARTPROPERTIES** followed by the number of properties that follow. If present, the number needs to match the number of lines following this one before the occurrence of a line beginning with **ENDPROPERTIES** These lines consist of a word for the property name followed by either an integer or string surrounded by double quotes. Properties named **FONT_ASCENT FONT_DESCENT** and DEFAULT_CHAR are typically present in BDF files to define the logical font-ascent and font-descent and the default-char for the font.

As mentioned above, this section, if it exists, must be terminated by **ENDPROPERTIES**.

The word **CHARS** followed by the number of characters that follow. This number should always be **256**.

This terminates the part of the **loadfont** input file describing features of the font in general. The rest of the file contains descriptions of the individual characters. They consist of the following parts:

The word **STARTCHAR** followed by up to 14 characters (no blanks) describing the character. This can either be something like **C0041**, which indicates the hex value of the character or **uppercaseA**, which describes the character.

The word **ENCODING** followed by a positive integer representing value by which this character is represented internally in the codeset for which this font is used. The integer needs to be specified in decimal.

The word **SWIDTH** followed by the scalable width in x and y of character. Scalable widths are in units of 1/1000th of the size of the character. The y value should always be **0**; the x value is typically *666* for the type of characters used with **loadfont**. The values are not checked by the **loadfont** utility, but this line needs to be there for compatibility purposes.

The word **DWIDTH** followed by two numbers, which in a **BDF** file would mean the width in x and y of the character in device units. The y value is always zero. The x value is typically 8. **loadfont** checks only for the presence of the **DWIDTH** keyword.

The word **BBX** followed by the width in x, height in y and x and y displacement of the lower left-hand corner from the origin of the character.

Most fonts used by video cards will not use the bottom 4 rows of pixels, which basically means a vertical (y) displacement of –4. The only width allowed by **loadfont** is 8; heights supported are 8, 14, and 16. All **BBX** lines of the subsequent characters should list the same height and width as the first one (because only fixed size fonts are supported).

The optional word **ATTRIBUTES** followed by the attributes as 4 hex-encoded characters. The **loadfont** utility will accept this line, if present, but there is no meaning attached to it.

The word **BITMAP**, which indicates the beginning of the bitmap representation of the character. This line should be followed by **height** number of lines (height as specified in the **BBX** line) representing a hex-encoded bitmap of the character, one byte per line.

The word **ENDCHAR** indicating the end of the bitmap for this character.

After all the bitmaps, the end of the file is indicated by the **ENDFONT** keyword.

Example

The following example lists the beginning of the **loadfont** input file for an 8 by 16 font, supporting the IBM 437 codeset, as well as the bitmap representation of the character uppercase A.

```
STARTFONT 2.1
FONT 8x16
SIZE 16 75 75
FONTBOUNDINGBOX 8 16 0 -4
STARTPROPERTIES 3
FONT_DESCENT 4
FONT_ASCENT 12
DEFAULT_CHAR 0
ENDPROPERTIES
CHARS 256
STARTCHAR C0000
ENCODING 0
...
```

Bitmap for uppercase A character:

```
STARTCHAR C0041
ENCODING 65
SWIDTH 666 0
DWIDTH 8 0
BBX 8 16 0 -4
BITMAP
00
00
10
38
6c
c6
c6
fe
c6
c6
c6
c6
00
00
00
00
ENDCHAR
```

FILES **/usr/share/lib/*.bdf**

SEE ALSO **loadfont**(1)

NAME

logindevperm, fbtab – login-based device permissions

SYNOPSIS

/etc/logindevperm

DESCRIPTION

The **/etc/logindevperm** file contains information that is used by **login**(1) and **ttymon**(1M) to change the owner, group, and permissions of devices upon logging into or out of a console device. By default, this file contains lines for the keyboard, mouse, audio, and frame buffer devices.

The owner of the devices listed in **/etc/logindevperm** is set to the owner of the console by **login**(1). The group of the devices is set to the owner's group specified in **/etc/passwd**. The permissions are set as specified in **/etc/logindevperm**.

Fields are separated by TAB and/or SPACE characters. Blank lines and comments can appear anywhere in the file; comments start with a hashmark, '#', and continue to the end of the line.

The first field specifies the name of a console device (for example, **/dev/console**). The second field specifies the permissions to which the devices in the *device_list* field (third field) will be set. A *device_list* is a colon-separated list of device names. A device entry that is a directory name and ends with "/*" specifies all entries in the directory (except "." and ".."). For example, "/dev/fbs/*" specifies all frame buffer devices.

Once the devices are owned by the user, their permissions and ownership can be changed using **chmod**(1) and **chown**(1), as with any other user-owned file.

Upon logout the owner and group of these devices will be reset by **ttymon**(1M) to owner **root** and root's group as specified in **/etc/passwd** (typically **other**). The permissions are set as specified in the **/etc/logindevperm** file.

FILES

/etc/passwd File that contains user group information.

SEE ALSO

chmod(1), **chown**(1), **login**(1), **ttymon**(1M), **passwd**(4)

NOTES

/etc/logindevperm provides a superset of the functionality provided by **/etc/fbtab** in SunOS 4.x releases.

NAME

loginlog – log of failed login attempts

DESCRIPTION

After five unsuccessful login attempts, all the attempts are logged in the file **/var/adm/loginlog**. This file contains one record for each failed attempt. Each record contains the login name, tty specification, and time.

This is an ASCII file. Each field within each entry is separated from the next by a colon. Each entry is separated from the next by a new-line.

By default, **loginlog** does not exist, so no logging is done. To enable logging, the log file must be created with read and write permission for owner only. Owner must be **root** and group must be **sys**.

FILES

/var/adm/loginlog

SEE ALSO

login(1), **passwd**(1)

NAME

magic – file command's magic number file

SYNOPSIS

/etc/magic

DESCRIPTION

The **file**(1) command identifies the type of a file using, among other tests, a test for whether the file begins with a certain *magic number*. The **/etc/magic** file specifies what magic numbers are to be tested for, what message to print if a particular magic number is found, and additional information to extract from the file.

Each line of the file specifies a test to be performed. A test compares the data starting at a particular offset in the file with a 1-byte, 2-byte, or 4-byte numeric value or a string. If the test succeeds, a message is printed. The line consists of the following fields:

offset type value message

offset A number specifying the offset, in bytes, into the file of the data which is to be tested.

type The type of the data to be tested. The possible values are:

byte A one-byte value.

short A two-byte value.

long A four-byte value.

string A string of bytes.

The types **byte**, **short**, and **long** may optionally be followed by a mask specifier of the form **&***number*. If a mask specifier is given, the value is AND'ed with the *number* before any comparisons are done. The *number* is specified in C form. For instance, **13** is decimal, **013** is octal, and **0x13** is hexadecimal.

value The value to be compared with the value from the file. If the type is numeric, this value is specified in **C** form. If it is a string, it is specified as a **C** string with the usual escapes permitted (for instance, \n for NEWLINE).

Numeric values may be preceded by a character indicating the operation to be performed. It may be '**=**', to specify that the value from the file must equal the specified value, '**<**', to specify that the value from the file must be less than the specified value, '**>**', to specify that the value from the file must be greater than the specified value, '**&**', to specify that all the bits in the specified value must be set in the value from the file, '**^**', to specify that at least one of the bits in the specified value must not be set in the value from the file, or **x** to specify that any value will match. If the character is omitted, it is assumed to be '**=**'.

For string values, the byte string from the file must match the specified byte string. The byte string from the file which is matched is the same length as the specified byte string.

message The message to be printed if the comparison succeeds. If the string contains a **printf**(3S) format specification, the value from the file (with any specified masking performed) is printed using the message as the format string.

Some file formats contain additional information which is to be printed along with the file type. A line which begins with the character '>' indicates additional tests and messages to be printed. If the test on the line preceding the first line with a '>' succeeds, the tests specified in all the subsequent lines beginning with '>' are performed, and the messages printed if the tests succeed. The next line which does not begin with a '>' terminates this.

FILES **/etc/magic**

SEE ALSO **file**(1), **file**(1B), **printf**(3S)

BUGS There should be more than one level of subtests, with the level indicated by the number of '>' at the beginning of the line.

NAME

mnttab – mounted file system table

DESCRIPTION

The file **mnttab** resides in **/etc** and contains information about devices that are *currently* mounted. **mnttab** is read by programs using the routines described in **getmntent**(3C). **mount**(1M) adds entries to this file. **umount** removes entries from this file. Each entry is a line of fields separated by spaces in the form:

special mount_point fstype options time

where

special	The name of the resource to be mounted.
mount_point	The pathname of the directory on which the filesystem is mounted.
fstype	The file system type of the mounted file system.
options	The mount options.
time	The time at which the file system was mounted.

Examples of entries for the *special* field include the pathname of a block-special device, the name of a remote filesystem in *host:pathname* form, or the name of a ‘‘swap file’’ (for instance, a file made with **mkfile**(1M)).

FILES

/etc/mnttab

SEE ALSO

mkfile(1M), **mount**(1M), **setmnt**(1M), **getmntent**(3C)

NAME

netconfig – network configuration database

SYNOPSIS

/etc/netconfig

DESCRIPTION

The network configuration database, **/etc/netconfig**, is a system file used to store information about networks that are connected to the system. The **netconfig** database and the routines that access it (see **getnetconfig**(3N)) are part of the Network Selection component. The Network Selection component also includes **getnetpath**(3N) routines to provide application-specific network search paths. These routines access the **netconfig** database based on the environment variable **NETPATH** (see **environ**(5)).

netconfig contains an entry for each network available on the system. Entries are separated by newlines. Fields are separated by whitespace and occur in the order in which they are described below. Whitespace can be embedded as ''*blank*'' or ''*tab*''. Backslashes may be embedded as ''\\''. Lines in **/etc/netconfig** that begin with a **#** (hash) in column 1 are treated as comments.

Each of the valid lines in the **netconfig** database correspond to an available transport. Each entry is of the form:

network ID semantics flag protocol-family protocol-name network-device translation-libraries

network ID A string used to uniquely identify a network. *network ID* consists of non-null characters, and has a length of at least 1. No maximum length is specified. This namespace is locally significant and the local system administrator is the naming authority. All *network ID*s on a system must be unique.

semantics The *semantics* field is a string identifying the ''semantics'' of the network, that is, the set of services it supports, by identifying the service interface it provides. The *semantics* field is mandatory. The following semantics are recognized.

tpi_clts Transport Provider Interface, connectionless

tpi_cots Transport Provider Interface, connection oriented

tpi_cots_ord Transport Provider Interface, connection oriented, supports orderly release.

flag The *flag* field records certain two-valued (''true'' and ''false'') attributes of networks. *flag* is a string composed of a combination of characters, each of which indicates the value of the corresponding attribute. If the character is present, the attribute is ''true.'' If the character is absent, the attribute is ''false.'' ''**-**'' indicates that none of the attributes are present. Only one character is currently recognized:

v Visible (''default'') network. Used when the environment variable **NETPATH** is unset.

protocol family

The *protocol family* and *protocol name* fields are provided for protocol-specific applications.

The *protocol family* field contains a string that identifies a protocol family. The *protocol family* identifier follows the same rules as those for *network ID*s; the string consists of non-null characters, it has a length of at least **1**, and there is no maximum length specified. A ''–'' in the *protocol family* field indicates that no protocol family identifier applies (the network is experimental). The following are examples:

loopback	Loopback (local to host).
inet	Internetwork: UDP, TCP, etc.
implink	ARPANET imp addresses
pup	PUP protocols: for example, BSP
chaos	MIT CHAOS protocols
ns	XEROX NS protocols
nbs	NBS protocols
ecma	European Computer Manufacturers Association
datakit	DATAKIT protocols
ccitt	CCITT protocols, X.25, etc.
sna	IBM SNA
decnet	DECNET
dli	Direct data link interface
lat	LAT
hylink	NSC Hyperchannel
appletalk	Apple Talk
nit	Network Interface Tap
ieee802	IEEE 802.2; also ISO 8802
osi	Umbrella for all families used by OSI (for example, protosw lookup)
x25	CCITT X.25 in particular
osinet	AFI = 47, IDI = 4
gosip	U.S. Government OSI

protocol name The *protocol name* field contains a string that identifies a protocol. The *protocol name* identifier follows the same rules as those for *network ID*s; that is, the string consists of non-NULL characters, it has a length of at least **1**, and there is no maximum length specified. A ''–'' indicates that none of the names listed apply. The following protocol names are recognized.

tcp	Transmission Control Protocol
udp	User Datagram Protocol
icmp	Internet Control Message Protocol

network device

The *network device* is the full pathname of the device used to connect to the transport provider. Typically, this device will be in the **/dev** directory. The *network device* must be specified.

translation libraries

The *name-to-address translation libraries* support a ''directory service'' (a name-to-address mapping service) for the network. A ''–'' in this field indicates the absence of any *translation libraries*. This has a special meaning for networks of the protocol family **inet :** its name-to-address mapping is provided by the name service switch based on the entries for **hosts** and **services** in **nsswitch.conf**(4). For networks of other families, a ''–'' indicates non-functional name-to-address mapping. Otherwise, this field consists of a comma-separated list of pathnames to dynamically linked libraries. The pathname of the library can be either absolute or relative. See **dlopen**(3X).

Each field corresponds to an element in the **struct netconfig** structure. **struct netconfig** and the identifiers described on this manual page are defined in **<netconfig.h>**. This structure includes the following members:

char **nc_netid*	Network ID, including NULL terminator.
unsigned long *nc_semantics*	Semantics.
unsigned long *nc_flag*	Flags.
char **nc_protofmly*	Protocol family.
char **nc_proto*	Protocol name.
char **nc_device*	Full pathname of the network device.
unsigned long *nc_nlookups*	Number of directory lookup libraries.
char ***nc_lookups*	Names of the name-to-address translation libraries.
unsigned long *nc_unused[9]*	Reserved for future expansion.

The *nc_semantics* field takes the following values, corresponding to the semantics identified above:

NC_TPI_CLTS
NC_TPI_COTS
NC_TPI_COTS_ORD

The *nc_flag* field is a bitfield. The following bit, corresponding to the attribute identified above, is currently recognized. **NC_NOFLAG** indicates the absence of any attributes.

NC_VISIBLE

EXAMPLES

Below is a sample **netconfig** file:

```
#
# The "Network Configuration" File.
#
# Each entry is of the form:
#
#  <network_id> <semantics> <flags> <protofamily> <protoname> <device> \
#       <nametoaddr_libs>
#
# The "-" in <nametoaddr_libs> for inet family transports indicates
# redirection to the name service switch policies for "hosts" and
# "services". The "-" may be replaced by nametoaddr libraries that
# comply with the SVr4 specs, in which case the name service switch
# will not be used for netdir_getbyname, netdir_getbyaddr,
# gethostbyname, gethostbyaddr, getservbyname, and getservbyport.
# There are no nametoaddr_libs for the inet family in Solaris anymore.
#
udp        tpi_clts      v  inet      udp  /dev/udp      -
tcp        tpi_cots_ord  v  inet      tcp  /dev/tcp      -
rawip      tpi_raw       -  inet      -    /dev/rawip    -
ticlts     tpi_clts      v  loopback  -    /dev/ticlts   straddr.so
ticotsord  tpi_cots_ord  v  loopback  -    /dev/ticotsord straddr.so
ticots     tpi_cots      v  loopback  -    /dev/ticots   straddr.so
```

FILES

<netconfig.h>

SEE ALSO

dlopen(3X), **getnetconfig**(3N), **getnetpath**(3N), **nsswitch.conf**(4)

Network Interfaces Programmer's Guide
File System Administration

NAME

netgroup – list of network groups

SYNOPSIS

/etc/netgroup

DESCRIPTION

A **netgroup** defines a network-wide group of hosts and users.

Netgroups may be used to restrict access to shared NFS filesystems and for restricting remote login and shell access.

Network groups are stored in one of the Network Information Services, either NIS or NIS+, not in a local file.

This manual page describes the format for a file that may be used to supply input to the **makedbm**(1M) or **nisaddent**(1M) programs that are use to build the NIS map or NIS+ table, respectively.

Each line of the file defines the name and membership of network group. The line should have the format:

groupname member ...

The items on a line may be separated by a combination of one or more spaces or tabs.

The *groupname* is the name of the group being defined. This is followed by a list of members of the group. Each *member* is either another group name, all of whose members are to be included in the group being defined, or a triple of the form:

(hostname,username,domainname)

In each triple, any of the three fields *hostname, username,* and *domainname,* can be empty. An empty field signifies a "wildcard" matching any value in that field. Thus:

everything (,,this.domain)

defines a group named "everything" for the domain "this.domain" to which every host and user belongs.

The *domainname* field refers to the domain in which the triple is valid, not the domain containing the host or user.

Netgroups can be used to control NFS mount access (see **share_nfs**(1M)) and to control remote login and shell access (see **hosts.equiv**(4)). They can also be used to control local login access (see **passwd**(4), **shadow**(4), and "compat" in **nsswitch.conf**(4)).

When used for these purposes, a host is considered a member of a netgroup if the netgroup contains any triple in which the *hostname* field matches the name of the host *requesting* access and the *domainname* field matches the domain of the host *controlling* access.

Similarly, a user is considered a member of a netgroup if the netgroup contains any triple in which the *username* field matches the name of the user requesting access and the *domainname* field matches the domain of the host controlling access.

Note that when netgroups are used to control NFS mount access, access is granted depending only on whether the requesting host is a member of the netgroup. Remote login and shell access can be controlled both on the basis of host and user membership in

separate netgroups.

FILES

/etc/netgroup used by **/var/yp/Makefile** on NIS masters to build the NIS netgroup map

Note that the netgroup information must always be stored in a network information service, either NIS or NIS+. The local file is only used to construct the netgroup NIS maps or NIS+ table; it is never consulted directly.

SEE ALSO

nis+(1), **makedbm**(1M), **nisaddent**(1M), **share_nfs**(1M), **innetgr**(3N), **hosts.equiv**(4), **nsswitch.conf**(4), **passwd**(4), **shadow**(4)

NOTES

Applications may make general membership tests using the **innetgr()** function (see **innetgr**(3N)).

Because the "-" character will not match any specific username or hostname, it is commonly used as a placeholder that will match only wildcarded membership queries. So, for example:

```
onlyhosts   (host1,-,our.domain) (host2,-,our.domain)
onlyusers   (-,john,our.domain) (-,linda,our.domain)
```

effectively define netgroups containing only hosts and only users, respectively. Any other string that is guaranteed not to be a legal username or hostname will also suffice for this purpose.

When a machine with multiple interfaces and multiple names is defined as a member of a netgroup, one must list all of the names (see **hosts**(4)). A manageable way to do this is to define a netgroup containing all of the machine names. For example, for a host "gateway" that has names "gateway-subnet1" and "gateway-subnet2" one may define the netgroup:

gateway (gateway-subnet1, ,our.domain) (gateway-subnet2, ,our.domain)

and use this netgroup **gateway** whenever the host is to be included in another netgroup.

NAME

netid – netname database

SYNOPSIS

/etc/netid

DESCRIPTION

The **netid** file is a local source of information on mappings between netnames (see **secure_rpc**(3N)) and user ids or hostnames in the local domain. The **netid** file can be used in conjunction with, or instead of, the network source: NIS or NIS+. The **publickey** entry in the **nsswitch.conf** (see **nsswitch.conf**(4)) file determines which of these sources will be queried by the system to translate netnames to local user ids or hostnames.

Each entry in the **netid** file is a single line of the form:

netname *uid*:*gid, gid, gid*...

or

netname **0:***hostname*

The first entry associates a local user id with a netname. The second entry associates a hostname with a netname.

The **netid** file field descriptions are as follows:

netname The operating system independent network name for the user or host. *netname* has one of two formats. The format used to specify a host is of the form:

unix.*hostname***@***domain*

where *hostname* is the name of the host and *domain* is the network domain name.

The format used to specify a user id is of the form:

unix.*uid***@***domain*

where *uid* is the numerical id of the user and *domain* is the network domain name.

uid The numerical id of the user (see **passwd**(4)). When specifying a host name, *uid* is always zero.

group The numerical id of the group the user belongs to (see **group**(4)). Several groups, separated by commas, may be listed for a single *uid*.

hostname The local hostname (see **hosts**(4)).

Blank lines are ignored. Any part of a line to the right of a '**#**' symbol is treated as a comment.

EXAMPLES

Here is a sample **netid** file:

```
unix.789@West.Sun.COM              789:30,65
unix.123@Bldg_xy.Sun.COM           123:20,1521
unix.candlestick@campus1.bayarea.EDU  0:candlestick
```

FILES

/etc/group	groups file
/etc/hosts	hosts database
/etc/netid	netname database
/etc/passwd	password file
/etc/publickey	public key database

SEE ALSO

netname2user(3N), **group**(4), **hosts**(4), **nsswitch.conf**(4), **passwd**(4), **publickey**(4)

NAME

netmasks – network mask database

SYNOPSIS

/etc/inet/netmasks
/etc/netmasks

DESCRIPTION

The **netmasks** file contains network masks used to implement IP standard subnetting. For each network that is subnetted, a single line should exist in this file with the network number, any number of SPACE or TAB characters, and the network mask to use on that network. Network numbers and masks may be specified in the conventional IP dot ('**.**') notation (like IP host addresses, but with zeroes for the host part). For example,

128.32.0.0 255.255.255.0

can be used to specify that the Class B network 128.32.0.0 should have eight bits of subnet field and eight bits of host field, in addition to the standard sixteen bits in the network field.

SEE ALSO

ifconfig(1M)

Postel, Jon, and Mogul, Jeff, *Internet Standard Subnetting Procedure*, RFC 950, Network Information Center, SRI International, Menlo Park, Calif., August 1985.

NOTES

/etc/inet/netmasks is the official SVR4 name of the **netmasks** file. The symbolic link **/etc/netmasks** exists for BSD compatibility.

NAME

netrc – file for ftp remote login data

DESCRIPTION

The **.netrc** file contains data for logging in to a remote host over the network for file transfers by **ftp**(1). This file resides in the user's home directory on the machine initiating the file transfer. Its permissions should be set to disallow read access by group and others (see **chmod**(1)).

The following tokens are recognized; they may be separated by SPACE, TAB, or NEWLINE characters:

machine *name*
: Identify a remote machine name. The auto-login process searches the **.netrc** file for a **machine** token that matches the remote machine specified on the **ftp** command line or as an **open** command argument. Once a match is made, the subsequent **.netrc** tokens are processed, stopping when the EOF is reached or another **machine** token is encountered.

login *name*
: Identify a user on the remote machine. If this token is present, the auto-login process will initiate a login using the specified name.

password *string*
: Supply a password. If this token is present, the auto-login process will supply the specified string if the remote server requires a password as part of the login process. Note: if this token is present in the **.netrc** file, **ftp** will abort the auto-login process if the **.netrc** is readable by anyone besides the user.

account *string*
: Supply an additional account password. If this token is present, the auto-login process will supply the specified string if the remote server requires an additional account password, or the auto-login process will initiate an **ACCT** command if it does not.

macdef *name*
: Define a macro. This token functions the same as **ftp macdef**. A macro is defined with the specified name; its contents begin with the next **.netrc** line and continue until a null line (consecutive NEWLINE characters) is encountered. If a macro named **init** is defined, it is automatically executed as the last step in the auto-login process.

EXAMPLES

A **.netrc** file containing the following line:

machine ray login demo password mypassword

allows an autologin to the machine **ray** using the login name **demo** with password **mypassword**.

FILES **~/.netrc**

SEE ALSO **chmod**(1), **ftp**(1), **in.ftpd**(1M)

NAME

networks – network name database

/etc/inet/networks
/etc/networks

DESCRIPTION

The **networks** file is a local source of information regarding the networks which comprise the Internet. The networks file can be used in conjunction with, or instead of, other networks sources, including the NIS maps **networks.byname** and **networks.byaddr** and the NIS+ table **networks**. Programs use the **getnetbyname**(3N) routines to access this information.

The network file has a single line for each network, with the following information:

official-network-name network-number aliases

Items are separated by any number of SPACE and/or TAB characters. A '**#**' indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file. This file is normally created from the official network database maintained at the Network Information Control Center (NIC), though local changes may be required to bring it up to date regarding unofficial aliases and/or unknown networks.

Network numbers may be specified in the conventional dot ('**.**') notation using the **inet_network** routine from the Internet address manipulation library, **inet**(7). Network names may contain any printable character other than a field delimiter, NEWLINE, or comment character.

SEE ALSO

getnetbyname(3N), **inet**(3N), **nsswitch.conf**(4), **inet**(7)

NOTES

/etc/inet/networks is the official SVR4 name of the **networks** file. The symbolic link **/etc/networks** exists for BSD compatibility.

NAME

nisfiles – NIS+ database files and directory structure

SYNOPSIS

/var/nis

DESCRIPTION

The Network Information Service Plus (NIS+) uses a memory based, replicated database. This database uses a set of files in the **/var/nis** directory for checkpointing to stable storage and for maintaining a transaction log. Additionally, the NIS+ server and client use files in this directory to store binding and state information.

The NIS+ service implements an authentication and authorization system that is built upon Secure RPC. In this implementation, the service uses a table named **cred.org_dir.***domain-name* to store the public and private keys of principals that are authorized to access the NIS+ namespace. It stores group access information in the sub-domain **groups_dir.***domain-name* as *group* objects. These two tables appear as files in the **/var/nis/***hostname* directory on the NIS+ server.

Unlike the previous versions of the network information service in NIS+, the information in the tables is initially loaded into the service from the ASCII files on the server and then updated using NIS+ utilities (**nistbladm** –**D**). Some sites may wish to periodically regenerate the ASCII files for archival purposes. To do this, a script should be added in the **crontab**(1) of the server that lists these tables and creates the ASCII file from the result.

Note: Except for the **NIS_COLDSTART** and **NIS_SHARED_DIRCACHE** file, no other files should be manipulated by commands such as **cp**(1), **mv**(1) or **rm**(1). The transaction log file keeps logs of all changes made, and hence the files cannot be manipulated independently.

The files described below are stored in the **/var/nis** directory:

NIS_COLDSTART
: This file contains NIS+ directory objects that are to be preloaded into the NIS+ cache at startup time. This file is usually created at NIS+ installation time. See **nisinit**(1M) or **nisclient**(1M).

NIS_SHARED_DIRCACHE
: This file contains the current cache of NIS+ bindings being maintained by the cache manager. The contents can be viewed with **nisshowcache**(1M).

*hostname***.log**
: This file contains a transaction log that is maintained by the NIS+ service. It can be viewed using the **nislog**(1M) command. This file contains holes. Its apparent size may be a lot higher than its actual size. There is only one transaction log per server.

*hostname***.dict**
: This file is a dictionary that is used by the NIS+ database to locate its files. It is created by the default NIS+ database package.

hostname.**dict.log**
: This is the log file for the database dictionary. When the server is checkpointed (**nisping -C**), this file will be deleted.

hostname This directory contains databases that the server uses.

hostname/**root.object**
On root servers, this file contains a directory object that describes the root of the name space.

hostname/**parent.object**
On root servers, this file contains a directory object that describes the parent namespace. This file is created by the **nisinit**(1M) command.

hostname/*table_name*
For each table in the directory there will be a file with the same name that stores the information about that table. If there are subdirectories within this directory, the database for the table is stored in the file *table_name*.*subdirectory*.

hostname/*table_name.log*
This file contains the database log for the table *table_name*. The log file maintains the state of individual transactions to each database. When a database has been checkpointed (that is, all changes have been made to the *hostname* / *table_name* stable storage), this log file will be deleted.

Currently, NIS+ does not automatically do checkpointing. The system administrator may want to do **nisping** –**C** (see **nisping**(1M)) operations periodically (such as, once a day) to checkpoint the log file. This can be done either through a **cron**(1M) job, or manually.

hostname/**root_dir**
On root servers, this file stores the database associated with the root directory. It is similar to other table databases. The corresponding log file is called **root_dir.log**.

hostname/**cred.org_dir**
This table contains the credentials of principals in this NIS+ domain.

hostname/**groups_dir**
This table contains the group authorization objects needed by NIS+ to authorize group access.

hostname/**serving_list**
This file contains a list of all NIS+ directories that are being served by the NIS+ server on this server. When this server is added or deleted from any NIS+ directory object, this file is updated by the server.

SEE ALSO

cp(1), **crontab**(1), **mv**(1), **rm**(1), **nis+**(1), **niscat**(1), **nismatch**(1), **nistbladm**(1), **nisclient**(1M), **nisinit**(1M), **nislog**(1M), **nisping**(1M), **nis_db**(3N), **nis_objects**(3N)

NAME

nsswitch.conf – configuration file for the name-service switch

SYNOPSIS

/etc/nsswitch.conf

DESCRIPTION

The operating system uses a number of "databases" of information about hosts, users (**passwd** / **shadow**), groups and so forth. Data for these can come from a variety of sources: host-names and -addresses, for example, may be found in **/etc/hosts**, NIS, NIS+ or DNS. One or more sources may be used for each database; the sources and their lookup order are specified in the **/etc/nsswitch.conf** file.

The following databases use the switch:

Database	*Used by*
aliases	**sendmail**(1M)
automount	**automount**(1M)
bootparams	**rpc.bootparamd**(1M)
ethers	**ethers**(3N)
group	**getgrnam**(3C)
hosts	**gethostbyname**(3N) *(See "Interaction with netconfig" below)*
netgroup	**innetgr**(3N)
netmasks	**ifconfig**(1M)
networks	**getnetbyname**(3N)
passwd	**getpwnam**(3C), **getspnam**(3C)
protocols	**getprotobyname**(3N)
publickey	**getpublickey**(3N)**, secure_rpc**(3N)
rpc	**getrpcbyname**(3N)
sendmailvars	**sendmail**(1M)
services	**getservbyname**(3N) *(See "Interaction with netconfig" below)*

The following sources may be used:

Source	*Uses*
files	**/etc/hosts**, **/etc/passwd**, **/etc/shadow** and so forth
nis	NIS (YP)
nisplus	NIS+
dns	Valid only for **hosts;** uses the Internet Domain Name Service.
compat	Valid only for **passwd** and **group;** implements "+" and "-". *(See "Interaction with +/- syntax" below)*

There is an entry in **/etc/nsswitch.conf** for each database. Typically these entries will be simple, like "protocols: files" or "networks: files nisplus". However, when multiple sources are specified it is sometimes necessary to define precisely the circumstances under which each source will be tried. A source can return one of the following codes:

Status	*Meaning*
SUCCESS	Requested database entry was found
UNAVAIL	Source is not responding or corrupted

NOTFOUND	Source responded "no such entry"
TRYAGAIN	Source is busy, might respond to retries

For each status code, two actions are possible:

Action	*Meaning*
continue	Try the next source in the list
return	Return now

The complete syntax of an entry is

```
<entry>    ::= <database> ":" [<source> [<criteria>]]* <source>
<criteria>  ::= "[" <criterion>+ "]"
<criterion> ::= <status> "=" <action>
<status>   ::= "success" | "notfound" | "unavail" | "tryagain"
<action>   ::= "return"  | "continue"
```

Each entry occupies a single line in the file. Lines that are blank, or that start with white space character are ignored. Everything on a line following a **#** character is also ignored; the **#** character can begin anywhere in a line, to be used to begin comments. The <database> and <source> names are case-sensitive, but <action> and <status> names are case-insensitive.

The library functions contain compiled-in default entries that are used if the appropriate entry in nsswitch.conf is absent or syntactically incorrect.

The default criteria are to continue on anything except SUCCESS; in other words, [SUCCESS=return NOTFOUND=continue UNAVAIL=continue TRYAGAIN=continue].

The default, or explicitly specified, criteria are meaningless following the last source in an entry; and are ignored since the action is always to return to the caller irrespective of the status code the source returns.

Interaction with netconfig

In order to ensure that they all return consistent results, **gethostbyname**(3N), **getservbyname**(3N), and **netdir_getbyname**(3N) functions are all implemented in terms of the same internal library function. This function obtains the system-wide source lookup policy for **hosts** and **services** based on the **inet** family entries in **netconfig**(4) and uses the switch entries only if the netconfig entries have a "-" in the last column for name-toaddr libraries. See the **NOTES** section in **gethostbyname**(3N) and **getservbyname**(3N) for details.

Interaction with NIS+ YP-compatibility Mode

The NIS+ server can be run in "YP-compatibility mode", where it handles NIS (YP) requests as well as NIS+ requests. In this case, the clients get much the same results (except for **getspnam**(3C)) from the "nis" source as from "nisplus"; however, "nisplus" is recommended instead of "nis".

Interaction with NIS (YP) server in DNS-forwarding Mode

The NIS (YP) server can be run in "DNS-forwarding mode", where it forwards lookup requests to DNS for host-names and -addresses that do not exist in its database. In this case, specifying "nis" as a source for "hosts" is sufficient to get DNS lookups; "dns" need not be specified explicitly as a source.

Since SunOS 5.3, the NIS+ server in "YP-compatibility mode" can also be run in "DNS-forwarding mode" (see **rpc.nisd**(1M)). Forwarding is effective only for requests originating from its YP clients; "hosts" policy on these clients should be configured appropriately.

Interaction with +/- syntax

Releases prior to SunOS 5.0 did not have the name-service switch but did allow the user some policy control. In **/etc/passwd** one could have entries of the form +*user* (include the specified user from NIS passwd.byname), -*user* (exclude the specified user) and + (include everything, except excluded users, from NIS passwd.byname). The desired behavior was often "everything in the file followed by everything in NIS", expressed by a solitary + at the end of **/etc/passwd**. The switch provides an alternative for this case ("passwd: files nis") that does not require + entries in **/etc/passwd** and **/etc/shadow** (the latter is a new addition to SunOS 5.0, see **shadow**(4)).

If this is not sufficient, the "compat" source provides full +/- semantics. It reads **/etc/passwd** for **getpwnam**(3C) functions and **/etc/shadow** for **getspnam**(3C) functions and, if it finds +/- entries, invokes an appropriate source. By default the source is "nis", but this may be overridden by specifying "nisplus" as the source for the pseudo-database **passwd_compat.**

Note that for every **/etc/passwd** entry, there should be a corresponding entry in the **/etc/shadow** file.

The **compat** source also provides full +/- semantics for **group;** the relevant pseudo-database is **group_compat.**

Useful Configurations

The compiled-in default entries for all databases use NIS (YP) as the enterprise level name-service and are identical to those in the default configuration of this file:

```
passwd:       files nis
group:        files nis
hosts:        nis [NOTFOUND=return] files
networks:     nis [NOTFOUND=return] files
protocols:    nis [NOTFOUND=return] files
rpc:          nis [NOTFOUND=return] files
ethers:       nis [NOTFOUND=return] files
netmasks:     nis [NOTFOUND=return] files
bootparams:   nis [NOTFOUND=return] files
publickey:    nis [NOTFOUND=return] files
netgroup:     nis
automount:    files nis
aliases:      files nis
services:     files nis
sendmailvars: files
```

The policy "nis [NOTFOUND=return] files" implies "if **nis** is UNAVAIL, continue on to **files**, and if **nis** returns NOTFOUND, return to the caller; in other words, treat **nis** as the authoritative source of information and try **files** only if **nis** is down." This, and other policies listed in the default configuration above, are identical to the hard-wired policies in

SunOS releases prior to 5.0.

If compatibility with the +/- syntax for passwd and group is required, simply modify the entries for **passwd** and **group** to:

```
passwd:     compat
group:      compat
```

If NIS+ is the enterprise level name-service, the default configuration should be modified to use **nisplus** instead of **nis** for every database on client machines. The file **/etc/nsswitch.nisplus** contains a sample configuration that can be copied to **/etc/nsswitch.conf** to set this policy.

If the use of +/- syntax is desired in conjunction with **nisplus,** use the following four entries:

```
passwd:     compat
passwd_compat:  nisplus
group:      compat
group_compat:   nisplus
```

In order to get information from the Internet Domain Name Service for hosts that are not listed in the enterprise level name-service, NIS+, use the following configuration and set up the **/etc/resolv.conf** file (see **resolv.conf**(4) for more details):

```
hosts:      nisplus dns [NOTFOUND=return] files
```

Enumeration -- getXXXent()

Many of the databases have enumeration functions: **passwd** has **getpwent(), hosts** has **gethostent(),** and so on. These were reasonable when the only source was **files** but often make little sense for hierarchically structured sources that contain large numbers of entries, much less for multiple sources. The interfaces are still provided and the implementations strive to provide reasonable results, but the data returned may be incomplete (enumeration for **hosts** is simply not supported by the **dns** source), inconsistent (if multiple sources are used), formatted in an unexpected fashion (for a host with a canonical name and three aliases, the **nisplus** source will return four hostents, and they may not be consecutive), or very expensive (enumerating a **passwd** database of 5000 users is probably a bad idea). Furthermore, multiple threads in the same process using the same reentrant enumeration function (**getXXXent_r()** are supported beginning with SunOS 5.3) share the same enumeration position; if they interleave calls, they will enumerate disjoint subsets of the same database.

In general the use of the enumeration functions is deprecated. In the case of **passwd, shadow** and **group,** it may sometimes be appropriate to use **fgetgrent()**, **fgetpwent()** and **fgetspent()** (see **getgrnam**(3C), **getpwnam**(3C), and **getspnam**(3C), respectively), which use only the **files** source.

FILES

A source named SSS is implemented by a shared object named nss_SSS.so.1 that resides in **/usr/lib**.

/etc/nsswitch.conf	configuration file
/usr/lib/nss_compat.so.1	implements "compat" source
/usr/lib/nss_dns.so.1	implements "dns" source
/usr/lib/nss_files.so.1	implements "files" source
/usr/lib/nss_nis.so.1	implements "nis" source
/usr/lib/nss_nisplus.so.1	implements "nisplus" source
/etc/netconfig	configuration file for **netdir**(3N) functions that redirects hosts / sevices policy to the switch
/etc/nsswitch.files	sample configuration file that uses "files" only
/etc/nsswitch.nis	sample configuration file that uses "files" and "nis"
/etc/nsswitch.nisplus	sample configuration file that uses "files" and "nisplus"

SEE ALSO

nis+(1), **automount**(1M), **ifconfig**(1M), **rpc.bootparamd**(1M), **rpc.nisd**(1M), **sendmail**(1M), **getgrnam**(3C), **getpwnam**(3C), **getspnam**(3C), **ethers**(3N), **gethostbyname**(3N), **getnetbyname**(3N), **getnetgrent**(3N), **getprotobyname**(3N), **getpublickey**(3N), **getrpcbyname**(3N), **getservbyname**(3N), **netdir**(3N), **secure_rpc**(3N), **netconfig**(4), **resolv.conf**(4), **ypfiles**(4)

NOTES

Within each process that uses **nsswitch.conf**, the entire file is read only once; if the file is later changed, the process will continue using the old configuration.

Programs that use the **getXXbyYY()** functions cannot be linked statically since the implementation of these functions requires dynamic linker functionality to access the shared objects **/usr/lib/nss_SSS.so.1** at run time.

The use of both **nis** and **nisplus** as sources for the same database is strongly discouraged since both the name-services are expected to store similar information and the lookups on the database may yield different results depending on which name-service is operational at the time of the request.

Misspelled names of sources and databases will be treated as legitimate names of (most likely nonexistent) sources and databases.

The following functions do *not* use the switch: **fgetgrent**(3C), **fgetpwent**(3C), **fgetspent**(3C), **getpw**(3C), **putpwent**(3C).

NAME

order – package installation order description file

DESCRIPTION

The package installation order file, **.order**, is an ASCII file specifying the order in which packages must be installed based on their prerequisite dependencies. Any package with prerequisite dependencies must be installed *after* any packages it lists as a prerequisite dependency in its **depend** file.

A **.order** file is required for the OS product. The **.order** file must reside in the top-level directory containing the product.

The ordering is specified as a list of package identifiers, from the first package to be installed to the last, one package identifier per line.

NOTES

The **depend** file supports *incompatible* and *reverse* dependencies. These dependency types are not recognized in the **order** file.

SEE ALSO

cdtoc(4), **clustertoc**(4), **depend**(4), **packagetoc**(4), **pkginfo**(4)

NAME

ott, .ott – FACE object architecture information

DESCRIPTION

The FACE object architecture stores information about object-types in an ASCII file named **.ott** (object type table) that is contained in each directory. This file describes all of the objects in that directory. Each line of the **.ott** file contains information about one object in pipe-separated fields. The fields are (in order):

name	the name of the actual system file.
dname	the name that should be displayed to the user, or a dot if it is the same as the name of the file.
description	the description of the object, or a dot if the description is the default (the same as object-type).
object-type	the FACE internal object type name.
flags	object specific flags.
mod time	the time that FACE last modified the object. The time is given as number of seconds since 1/1/1970, and is in hexadecimal notation.
object information	an optional field, contains a set of semi-colon separated *name*=*value* fields that can be used by FACE to store any other information necessary to describe this object.

FILES

.ott is created in any directory opened by FACE.

NAME

packagetoc – package table of contents description file

DESCRIPTION

The package table of contents file, **.packagetoc,** is an ASCII file containing all of the information necessary for installing a product release distributed in package form. It centralizes and summarizes all of the relevant information about each package in the product. This allows the install software to quickly read one file to obtain all of the relevant information about each package instead of having to examine each package at run time to obtain this information. The **.packagetoc** file resides in the top-level directory containing the product.

If a **.packagetoc** file exists for a product, there must also be a **.order** file.

Each entry in the **.packagetoc** file is a line that establishes the value of a parameter in the following form:

PARAM=*value*

A line starting with a pound-sign, ''#'', is considered a comment and is ignored.

Parameters are grouped by package. The start of a package description is defined by a line of the form:

PKG=*value*

There is no order implied or assumed for specifying the parameters for a package with the exception of the **PKG** parameter, which must appear first. Only one occurrence of a parameter is permitted per package.

The parameters recognized are described below. Those marked with an asterisk are mandatory.

PKG*
: The package identifier (for example, **SUNWaccu**). The maximum length of the identifier is nine characters. All the characters must be alphanumeric. The first character must be alphabetic. **install**, **new**, and **all** are reserved identifiers.

PKGDIR*
: The name of the directory containing the package. This directory is relative to the directory containing the product.

NAME*
: The full name of the package.

VENDOR
: The name of the package's vendor.

VERSION
: The version of the package.

PRODNAME
: The name of the product to which this package belongs.

PRODVERS
: The version of the product to which this package belongs.

SUNW_PKGTYPE
: The package type. Valid values are:

  root indicates that the package will be installed in the **/** file system. The **root** packages are the only packages installed during dataless client installations. The **root** packages are spooled during a server installation to allow the later installation of diskless clients.

  usr indicates that the package will be installed in the **/usr** file

system.

kvm indicates that the package will be installed in the **/usr/kvm** file system.

ow indicates a package that is part of the bundled OpenWindows product release. If no **SUNW_PKGTYPE** macro is present, the package is assumed to be of type **usr**.

ARCH* The architecture(s) supported by the package. This macro is taken from the package's **pkginfo**(4) file and is subject to the same length and formatting constraints.

The install program currently assumes that exactly one architecture token is specified for a package. For example, **ARCH=sparc.sun4c** is acceptable, but **ARCH=sparc.sun4c, sparc.sun4m** is not.

DESC A detailed textual description of the package.

BASEDIR* The default installation base directory of the package.

SUNW_PDEPEND A dependency specification for a prerequisite package. Each prerequisite dependency must appear as a separate macro. See **depend**(4) for more information on dependencies and instance specifications.

SUNW_IDEPEND A dependency specification for an incompatible package. Each incompatible dependency should appear as a separate macro. See **depend**(4) for more information on dependencies and instance specifications.

SUNW_RDEPEND A dependency specification for a reversed package dependency. Each reverse dependency should appear as a separate macro. See **depend**(4) for more information on dependencies and instance specifications.

CATEGORY The category of the package.

SUNW_LOC Indicates that this package contains localizations for other packages. Such localization packages are treated as special case packages. Each package which has a **SUNW_LOC** macro must have a corresponding **SUNW_PKGLIST** macro. The value specified by this macro should be a valid locale.

SUNW_PKGLIST A comma separate list of package identifiers. Currently this macro is used to indicate which packages are localized by a localization package.

ROOTSIZE* The space used by the package in the **/** file system.

USRSIZE* The space used by the package in the **/usr** subtree of the file system.

VARSIZE* The space used by the package in the **/var** subtree of the file system.

OPTSIZE* The space used by the package in the **/opt** subtree of the file system.

EXPORTSIZE* The space used by the package in the **/export** subtree of the file system.

USROWNSIZE* The space used by the package in the **/usr/openwin** subtree of the file

system.

SPOOLED_SIZE* The space used by the spooled version of this package. This is used during the setup of a server by the initial system installation programs.

All sizes are specified in bytes. Default disk partitions and file system sizes are derived from the values provided: accuracy is important.

EXAMPLES

The following is an example package entry in a **.packagetoc** file.

```
#ident "@(#)packagetoc.4 1.2 92/04/28"
PKG=SUNWaccr
PKGDIR=SUNWaccr
NAME=System Accounting, (Root)
VENDOR=Sun Microsystems, Inc.
VERSION=8.1
PRODNAME=SunOS
PRODVERS=5.0beta2
SUNW_PKGTYPE=root
ARCH=sparc
DESC=System Accounting, (Root)
BASEDIR=/
CATEGORY=system
ROOTSIZE= 11264
VARSIZE= 15360
OPTSIZE= 0
EXPORTSIZE= 0
USRSIZE= 0
USROWNSIZE= 0
```

SEE ALSO

cdtoc(4), **clustertoc**(4), **depend**(4), **order**(4), **pkginfo**(4), **pkgmap**(4)

NOTES

The parameters **NAME**, **VENDOR**, **VERSION**, **PRODNAME**, **PRODVERS**, **SUNW_PKGTYPE**, **SUNW_LOC**, **SUNW_PKGLIST**, **ARCH**, **DESC**, **BASEDIR**, and **CATEGORY** are assumed to have been taken directly from the package's **pkginfo**(4) file. The length and formatting restrictions placed on the values for these parameters are identical to those for the corresponding entries in the **pkginfo**(4) file.

The value specified for the parameter **PKGDIR** should not exceed 255 characters.

The value specified for the parameters **ROOTSIZE**, **VARSIZE**, **OPTSIZE**, **EXPORTSIZE**, **USRSIZE** and **USROWNSIZE** must be a single integer value. The values can be derived from the package's **pkgmap** file by counting all space consumed by any files installed in the applicable file system. The space includes that used for directory entries and any UFS overhead that exists because of the way the files are represented (directory allocation scheme; direct, indirect, double indirect blocks; fragments; etc.)

The following kinds of entries in the **pkgmap**(4) file should be included in the space derivation:

f regular file
c character special file
b block special file
p pipe
l hard link
s symbolic link
x, d directory
i packaging installation script or information file (*copyright*, *depend*, *postinstall*, *postremove*)

NAME

passwd – password file

SYNOPSIS

/etc/passwd

DESCRIPTION

/etc/passwd is a local source of information about users' accounts. The password file can be used in conjunction with other password sources, including the NIS maps **passwd.byname** and **passwd.bygid** and the NIS+ table **passwd**. Programs use the **getpwnam**(3C) routines to access this information.

Each **passwd** entry is a single line of the form:

username:*password*:*uid*:*gid*:*gcos-field*:*home-dir*:*login-shell*

where

username is the user's login name. This field contains no uppercase characters, and must not be more than eight characters in length.

password is an empty field; The encrypted password for the user is in the corresponding entry in the **/etc/shadow** file. **pwconv**(1M) relies on a special value of '**x**' in the password field of **/etc/passwd**. If this value of '**x**' exists in the password field of **/etc/passwd**, this indicates that the password for the user is already in **/etc/shadow** and should not be modified.

uid is the user's unique numerical ID for the system.

gid is the unique numerical ID of the group that the user belongs to.

gcos-field is the user's real name, along with information to pass along in a mail-message heading. (It is called the gcos-field for historical reasons.) A ''**&**'' (ampersand) in this field stands for the login name (in cases where the login name appears in a user's real name).

home-dir is the pathname to the directory in which the user is initially positioned upon logging in.

login-shell is the user's initial shell program. If this field is empty, the default shell is **/usr/bin/sh**.

The password file is an ASCII file. Because the encrypted passwords are always kept in the shadow file, **/etc/passwd** has general read permission on all systems, and can be used by routines that map between numerical user IDs and user names.

Previous releases used a password entry beginning with a '+' (plus sign) or '–' (minus sign) to selectively incorporate entries from NIS maps for password. If still required, this is supported by specifying ''passwd : compat'' in **nsswitch.conf**(4). The ''compat'' source may not be supported in future releases. The preferred sources are, ''files'' followed by ''nisplus''. This has the effect of incorporating the entire contents of the NIS+ **passwd** table after the password file.

EXAMPLES

Here is a sample **passwd** file:

```
root:q.mJzTnu8icF.:0:10:God:/:/bin/csh
fred:6k/7KCFRPNVXg:508:10:% Fredericks:/usr2/fred:/bin/csh
```

and the sample password entry from **nsswitch.conf**:

```
passwd: files nisplus
```

In this example, there are specific entries for users **root** and **fred** to assure that they can login even when the system is running single-user. In addition, anyone in the NIS+ table **passwd** will be able to login with their usual password, shell and home directory.

If the password file is:

```
root:q.mJzTnu8icF.:0:10:God:/:/bin/csh
fred:6k/7KCFRPNVXg:508:10:% Fredericks:/usr2/fred:/bin/csh
+
```

and the password entry from nsswitch.conf:

```
passwd: compat
```

all the entries listed in the NIS **passwd.byuid** and **passwd.byname** maps will be effectively incorporated after the entries for **root** and **fred**.

FILES

/etc/nsswitch.conf
/etc/passwd
/etc/shadow

SEE ALSO

chgrp(1), **chown**(1), **groups**(1), **login**(1), **makekey**(1), **nispasswd**(1), **passwd**(1), **sh**(1), **sort**(1), **chown**(1M), **domainname**(1M), **getent**(1M), **in.ftpd**(1M), **newgrp**(1M), **passmgmt**(1M), **pwck**(1M), **pwconv**(1M), **su**(1M), **useradd**(1M), **userdel**(1M), **usermod**(1M), **a64l**(3C), **crypt**(3C), **getpw**(3C), **getpwnam**(3C), **getspnam**(3C), **putpwent**(3C), **group**(4), **hosts.equiv**(4), **nsswitch.conf**(4), **shadow**(4), **unistd**(4), **environ**(5)

NAME

path_to_inst – device instance number file

SYNOPSIS

/etc/path_to_inst

DESCRIPTION

/etc/path_to_inst records mappings of physical device names to instance numbers.

The instance number of a device is encoded in its minor number, and is the way that a device driver determines which of the possible devices that it may drive is referred to by a given special file.

In order to keep instance numbers persistent across reboots, the system records them in **/etc/path_to_inst**.

This file is read only at boot time, and is updated by **add_drv**(1M) and **drvconfig**(1M).

Note that it is generally not necessary for the system administrator to change this file, as the system will maintain it.

The system administrator can change the assignment of instance numbers by editing this file and doing a reconfiguration reboot. However, any changes made in this file will be lost if **add_drv**(1M) or **drvconfig**(1M) is run before the system is rebooted.

Each instance entry is a single line of the form:

"*physical name*" *instance number*

where

physical name is the physical pathname of a device. This pathname must be enclosed in " characters and start with /.

instance number is a decimal or hexadecimal number.

EXAMPLES

Here are some sample **path_to_inst** entries for a sun4c:

```
"/fd@1,f7200000" 0
"/audio@1,f7201000" 0
"/sbus@1,f8000000/esp@0,800000/sd@0,0" 0x0
"/sbus@1,f8000000/esp@0,800000/sd@1,0" 0x1
"/sbus@1,f8000000/esp@0,800000/sd@2,0" 0x2
"/sbus@1,f8000000/esp@0,800000/sd@3,0" 0x3
"/sbus@1,f8000000/le@0,c00000" 0
```

FILES

/etc/path_to_inst

SEE ALSO

add_drv(1M), **boot**(1M), **drvconfig**(1M), **mknod**(1M)

WARNING

If the file is removed the system may not be bootable (as it may rely on information found in this file to find the root, usr or swap device).If it does successfully boot it will regenerate the file, but after rebooting devices may end up having different minor numbers than they did before, and special files created via **mknod**(1M) may refer to different devices than expected.

For the same reasons, changes should not be made to this file without careful consideration.

NOTES

This document does not constitute an API. **path_to_inst** may not exist or may have a different content or interpretation in a future release. The existence of this notice does not imply that any other documentation that lacks this notice constitutes an API.

NAME

pathalias – alias file for FACE

SYNOPSIS

/usr/vmsys/pathalias

DESCRIPTION

The **pathalias** files contain lines of the form **alias=***path* where *path* can be one or more colon-separated directories. Whenever a FACE (Framed Access Command Environment, see **face**(1)) user references a path not beginning with a ‘‘/’’, this file is checked. If the first component of the pathname matches the left-hand side of the equals sign, the right-hand side is searched much like **$PATH** variable in the system. This allows users to reference the folder **$HOME/FILECABINET** by typing **filecabinet**.

There is a system-wide **pathalias** file called **$VMSYS/pathalias**, and each user can also have local alias file called **$HOME/pref/pathalias**. Settings in the user alias file override settings in the system-wide file. The system-wide file is shipped with several standard FACE aliases, such as **filecabinet**, **wastebasket**, **preferences**, **other_users**, etc.

FILES

$HOME/pref/pathalias
$VMSYS/pathalias

SEE ALSO

face(1)

NOTES

Unlike command keywords, partial matching of a path alias is not permitted, however, path aliases are case insensitive. The name of an alias should be alphabetic, and in no case can it contain special characters like ‘‘/’’, ‘‘\’’, or ‘‘=’’. There is no particular limit on the number of aliases allowed. Alias files are read once, at login, and are held in core until logout. Thus, if an alias file is modified during a session, the change will not take effect until the next session.

NAME

phones – remote host phone number database

SYNOPSIS

/etc/phones

DESCRIPTION

The file **/etc/phones** contains the system-wide private phone numbers for the **tip**(1) program. **/etc/phones** is normally unreadable, and so may contain privileged information. The format of **/etc/phones** is a series of lines of the form:

<*system-name*>**[\t]***<*phone-number*>.

The system name is one of those defined in the **remote**(4) file and the phone number is constructed from **[0123456789**–**=*%]**. The '**=**' and '*' characters are indicators to the auto call units to pause and wait for a second dial tone (when going through an exchange). The '**=**' is required by the DF02-AC and the '*' is required by the BIZCOMP 1030.

Comment lines are lines containing a '**#**' sign in the first column of the line.

Only one phone number per line is permitted. However, if more than one line in the file contains the same system name **tip**(1) will attempt to dial each one in turn, until it establishes a connection.

FILES

/etc/phones

SEE ALSO

tip(1), **remote**(4)

NAME

pkginfo – package characteristics file

DESCRIPTION

pkginfo is an ASCII file that describes the characteristics of the package along with information that helps control the flow of installation. It is created by the software package developer.

Each entry in the **pkginfo** file is a line that establishes the value of a parameter in the following form:

PARAM="*value*"

There is no required order in which the parameters must be specified within the file. Each parameter is described below. Only fields marked with an asterisk are mandatory.

*PKG** Abbreviation for the package being installed, generally three characters in length (for example, **dir** or **pkg**). All characters in the abbreviation must be alphanumeric and the first may not be numeric. The abbreviation is limited to a maximum length of nine characters. **install**, **new**, and **all** are reserved abbreviations.

*NAME** Text that specifies the package name (maximum length of 256 ASCII characters).

*ARCH** A comma-separated list of alphanumeric tokens that indicate the architecture associated with the package. The **pkgmk**(1) tool may be used to create or modify this value when actually building the package. The maximum length of a token is 16 characters and it cannot include a comma.

*VERSION** Text that specifies the current version associated with the software package. The maximum length is 256 ASCII characters and the first character cannot be a left parenthesis. The **pkgmk**(1) tool may be used to create or modify this value when actually building the package.

*CATEGORY** A comma-separated list of categories under which a package may be displayed. A package must at least belong to the system or application category. Categories are case-insensitive and may contain only alphanumerics. Each category is limited in length to 16 characters.

DESC Text that describes the package (maximum length of 256 ASCII characters).

VENDOR Used to identify the vendor that holds the software copyright (maximum length of 256 ASCII characters).

HOTLINE Phone number and/or mailing address where further information may be received or bugs may be reported (maximum length of 256 ASCII characters).

EMAIL	An electronic address where further information is available or bugs may be reported (maximum length of 256 ASCII characters).
VSTOCK	The vendor stock number, if any, that identifies this product (maximum length of 256 ASCII characters).
CLASSES	A space-separated list of classes defined for a package. The order of the list determines the order in which the classes are installed. Classes listed first will be installed first (on a media by media basis). This parameter may be modified by the request script.
ISTATES	A list of allowable run states for package installation (for example, "**S s 1**").
RSTATES	A list of allowable run states for package removal (for example, "**S s 1**").
BASEDIR	The pathname to a default directory where ''relocatable'' files may be installed. If blank, the package is not relocatable and any files that have relative pathnames will not be installed. An administrator can override the default directory.
ULIMIT	If set, this parameter is passed as an argument to the **ulimit** command, which establishes the maximum size of a file during installation.
ORDER	A list of classes defining the order in which they should be put on the medium. Used by **pkgmk**(1) in creating the package. Classes not defined in this field are placed on the medium using the standard ordering procedures.
MAXINST	The maximum number of package instances that should be allowed on a machine at the same time. By default, only one instance of a package is allowed. This parameter must be set in order to have multiple instances of a package.
PSTAMP	Production stamp used to mark the **pkgmap**(4) file on the output volumes. Provides a means for distinguishing between production copies of a version if more than one is in use at a time. If *PSTAMP* is not defined, the default is used. The default consists of the UNIX system machine name followed by the string "*YYMMDDHHMM*" (year, month, date, hour, minutes).
INTONLY	Indicates that the package should only be installed interactively when set to any non-NULL value.

EXAMPLES

Here is a sample **pkginfo**:

```
PKG="oam"
NAME="OAM Installation Utilities"
VERSION="3"
VENDOR="AT&T"
HOTLINE="1-800-ATT-BUGS"
EMAIL="attunix!olsen"
VSTOCK="0122c3f5566"
CATEGORY="system.essential"
ISTATES="S 2"
RSTATES="S 2"
```

SEE ALSO

pkgmap(4), **pkgmk**(1)

NOTES

Developers may define their own installation parameters by adding a definition to this file. A developer-defined parameter must begin with a capital letter.

NAME

pkgmap – package contents description file

DESCRIPTION

pkgmap is an ASCII file that provides a complete listing of the package contents. It is automatically generated by **pkgmk**(1) using the information in the **prototype** file.

Each entry in **pkgmap** describes a single ''deliverable object file.'' A deliverable object file includes shell scripts, executable objects, data files, directories, etc. The entry consists of several fields of information, each field separated by a space. The fields are described below and must appear in the order shown.

part An optional field designating the part number in which the object resides. A part is a collection of files, and is the atomic unit by which a package is processed. A developer can choose the criteria for grouping files into a part (for example, based on class). If no value is defined in this field, part 1 is assumed.

ftype A one-character field that indicates the file type. Valid values are:

- **f** a standard executable or data file
- **e** a file to be edited upon installation or removal
- **v** volatile file (one whose contents are expected to change)
- **d** directory
- **x** an exclusive directory
- **l** linked file
- **p** named pipe
- **c** character special device
- **b** block special device
- **i** installation script or information file
- **s** symbolic link

class The installation class to which the file belongs. This name must contain only alphanumeric characters and be no longer than 12 characters. It is not specified if the **ftype** is **i** (information file).

pathname *pathname* may contain variables that support install-time configuration of the file. A *$parameter* may be embedded in the pathname structure. Default values for *parameter* must be available in the environment during installation. The recommended method for setting such parameters is to supply them in the **pkginfo** file. Do not use the following reserved words in the **pkgmap** path, since they are applied by **pkgadd** using a different mechanism:

PKG_INSTALL_ROOT
BASEDIR
CLIENT_BASEDIR

major The major device number. The field is only specified for block or character special devices.

minor The minor device number. The field is only specified for block or character special devices.

mode The octal mode of the file (for example, 0664). A question mark (?) indicates that the mode will be left unchanged, implying that the file already exists on

the target machine. This field is not used for linked files, packaging information files or non-installable files.

owner The owner of the file (for example, **bin** or **root**). The field is limited to 14 characters in length. A question mark (**?**) indicates that the owner will be left unchanged, implying that the file already exists on the target machine. This field is not used for linked files or non-installable files. It is used optionally with a package information file. If used, it indicates with what owner an installation script will be executed.

Can be a variable specification in the form of $[A-Z]. Will be resolved at installation time.

group The group to which the file belongs (for example, "bin" or "sys"). The field is limited to 14 characters in length. A question mark (?) indicates that the group will be left unchanged, implying that the file already exists on the target machine. This field is not used for linked files or non-installable files. It is used optionally with a package information file. If used, it indicates with what group an installation script will be executed.

Can be a variable assignment in the form of **$[A-Z]**. Will be resolved at installation time.

size The actual size of the file in bytes. This field is not specified for named pipes, special devices, directories or linked files.

cksum The checksum of the file contents. This field is not specified for named pipes, special devices, directories or linked files.

modtime The time of last modification, as reported by the **stat**(2) function call. This field is not specified for named pipes, special devices, directories or linked files.

Each **pkgmap** must have one line that provides information about the number and maximum size (in 512-byte blocks) of parts that make up the package. This line is in the following format:

:*number_of_parts maximum_part_size*

Lines that begin with ‘‘**#**’’ are comment lines and are ignored.

When files are saved during installation before they are overwritten, they are normally just copied to a temporary pathname. However, for files whose mode includes execute permission (but which are not editable), the existing version is linked to a temporary pathname and the original file is removed. This allows processes which are executing during installation to be overwritten.

EXAMPLES

The following is an example of a **pkgmap** file.

```
:2 500
1 i pkginfo 237 1179 541296672
1 b class1 /dev/diskette 17 134 0644 root other
1 c class1 /dev/rdiskette 17 134 0644 root other
1 d none bin 0755 root bin
1 f none bin/INSTALL 0755 root bin 11103 17954 541295535
1 f none bin/REMOVE 0755 root bin 3214 50237 541295541
1 l none bin/UNINSTALL=bin/REMOVE
1 f none bin/cmda 0755 root bin 3580 60325 541295567
1 f none bin/cmdb 0755 root bin 49107 51255 541438368
1 f class1 bin/cmdc 0755 root bin 45599 26048 541295599
1 f class1 bin/cmdd 0755 root bin 4648 8473 541461238
1 f none bin/cmde 0755 root bin 40501 1264 541295622
1 f class2 bin/cmdf 0755 root bin 2345 35889 541295574
1 f none bin/cmdg 0755 root bin 41185 47653 541461242
2 d class2 data 0755 root bin
2 p class1 data/apipe 0755 root other
2 d none log 0755 root bin
2 v none log/logfile 0755 root bin 41815 47563 541461333
2 d none save 0755 root bin
2 d none spool 0755 root bin
2 d none tmp 0755 root bin
```

SEE ALSO

pkgmk(1)

NOTES

The **pkgmap** file may contain only one entry per unique pathname.

NAME

plot – graphics interface

DESCRIPTION

Files of this format are interpreted for various devices by commands described in **plot**(1B). A graphics file is a stream of plotting instructions. Each instruction consists of an ASCII letter usually followed by bytes of binary information. The instructions are executed in order. A point is designated by four bytes representing the *x* and *y* values; each value is a signed integer. The last designated point in an **l**, **m**, **n**, or **p** instruction becomes the ''current point'' for the next instruction.

m Move: the next four bytes give a new current point.

n Cont: draw a line from the current point to the point given by the next four bytes. See **plot**(1B).

p Point: plot the point given by the next four bytes.

l Line: draw a line from the point given by the next four bytes to the point given by the following four bytes.

t Label: place the following ASCII string so that its first character falls on the current point. The string is terminated by a NEWLINE.

a Arc: the first four bytes give the center, the next four give the starting point, and the last four give the end point of a circular arc. The least significant coordinate of the end point is used only to determine the quadrant. The arc is drawn counter-clockwise.

c Circle: the first four bytes give the center of the circle, the next two the radius.

e Erase: start another frame of output.

f Linemod: take the following string, up to a NEWLINE, as the style for drawing further lines. The styles are ''dotted,'' ''solid,'' ''longdashed,'' ''shortdashed,'' and ''dotdashed.'' Effective only in **plot 4014** and **plot ver**.

s Space: the next four bytes give the lower left corner of the plotting area; the following four give the upper right corner. The plot will be magnified or reduced to fit the device as closely as possible.

Space settings that exactly fill the plotting area with unity scaling appear below for devices supported by the filters of **plot**(1B). The upper limit is just outside the plotting area.

In every case the plotting area is taken to be square; points outside may be displayable on devices whose face is not square.

4014	**space(0, 0, 3120, 3120);**
ver	**space(0, 0, 2048, 2048);**
300, **300s**	**space(0, 0, 4096, 4096);**
450	**space(0, 0, 4096, 4096);**

SEE ALSO

graph(1), **plot**(1B)

NAME

proc, /proc – process file system

DESCRIPTION

/proc is a file system that provides access to the image of each process in the system. The name of each entry in the **/proc** directory is a decimal number corresponding to the process-ID. The owner of each ''file'' is determined by the process's real user-ID.

Standard system call interfaces are used to access **/proc** files: **open**(2), **close**(2), **read**(2), **write**(2), and **ioctl**(2). An open for reading and writing enables process control; a read-only open allows inspection but not control. As with ordinary files, more than one process can open the same **/proc** file at the same time. *Exclusive open* is provided to allow controlling processes to avoid collisions: an **open**(2) for writing that specifies O_EXCL fails if the file is already open for writing; if such an exclusive open succeeds, subsequent attempts to open the file for writing, with or without the O_EXCL flag, fail until the exclusively-opened file descriptor is closed. (Exception: a super-user **open**(2) that does not specify O_EXCL succeeds even if the file is exclusively opened.) There can be any number of read-only opens, even when an exclusive write open is in effect on the file.

Data may be transferred from or to any locations in the traced process's address space by applying **lseek**(2) to position the file at the virtual address of interest followed by **read**(2) or **write**(2). The **PIOCMAP** operation can be applied to determine the accessible areas (mappings) of the address space. I/O transfers may span contiguous mappings. An I/O request extending into an unmapped area is truncated at the boundary. An I/O request beginning at an unmapped virtual address fails with **EIO**.

Information and control operations are provided through **ioctl**(2). These have the form:

```
#include <sys/types.h>
#include <sys/signal.h>
#include <sys/fault.h>
#include <sys/syscall.h>
#include <sys/procfs.h>
void *p;
retval = ioctl(fildes, code, p);
```

The argument *p* is a generic pointer whose type depends on the specific **ioctl** code. Where not specifically mentioned below, its value should be zero. **<sys/procfs.h>** contains definitions of **ioctl** codes and data structures used by the operations.

Every active process contains at least one *light-weight process*, or *lwp*. Each *lwp* represents a flow of execution that is independently scheduled by the operating system. The **PIOCOPENLWP** operation can be applied to the process file descriptor to obtain a specific *lwp* file descriptor. I/O operations produce identical results whether applied to the process file descriptor or to an *lwp file descriptor*. All **/proc ioctl** operations can be applied to either type of file descriptor and, where not stated otherwise, produce identical results.

Process information and control operations involve the use of sets of flags. The set types **sigset_t**, **fltset_t**, and **sysset_t** correspond, respectively, to signal, fault, and system call enumerations defined in **<sys/signal.h>**, **<sys/fault.h>**, and **<sys/syscall.h>**. Each set type is large enough to hold flags for its own enumeration. Although they are of different sizes, they have a common structure and can be manipulated by these macros:

```
prfillset(&set);              /* turn on all flags in set */
premptyset(&set);             /* turn off all flags in set */
praddset(&set, flag);         /* turn on the specified flag */
prdelset(&set, flag);         /* turn off the specified flag */
r = prismember(&set, flag);   /* != 0 iff flag is turned on */
```

One of **prfillset( )** or **premptyset( )** must be used to initialize **set** before it is used in any other operation. **flag** must be a member of the enumeration corresponding to **set**.

IOCTLS

The allowable **ioctl** codes follow. Certain of these can be used only if the process or *lwp* file descriptor is open for writing; these include all operations that affect process control. Those requiring write access are marked with an asterisk (*). Except where noted, an **ioctl** to a process or *lwp* that has terminated elicits the error **ENOENT**.

PIOCSTATUS

PIOCSTATUS returns status information for the process and one of its *lwp*s; *p* is a pointer to a **prstatus** structure containing at least the following fields:

```
typedef struct prstatus {
    long            pr_flags;          /* Flags */
    short           pr_why;            /* Reason for stop (if stopped) */
    short           pr_what;           /* More detailed reason */
    id_t            pr_who;            /* Specific lwp identifier */
    u_short         pr_nlwp;           /* Number of lwps in the process */
    short           pr_cursig;         /* Current signal */
    sigset_t        pr_sigpend;        /* Set of process pending signals */
    sigset_t        pr_lwppend;        /* Set of lwp pending signals */
    sigset_t        pr_sighold;        /* Set of lwp held signals */
    struct siginfo  pr_info;           /* Info associated with signal */
                                       /* fault */
    struct sigaltstack pr_altstack;    /* Alternate signal stack info */
    struct sigaction pr_action;        /* Signal action for current signal */
    struct ucontext *pr_oldcontext;    /* Address of previous ucontext */
    caddr_t         pr_brkbase;        /* Address of the process heap */
    u_long          pr_brksize;        /* Size of the process heap, in bytes */
    caddr_t         pr_stkbase;        /* Address of the process stack */
    u_long          pr_stksize;        /* Size of the process stack, in */
                                       /* bytes */
    short           pr_syscall;        /* System call number (if in syscall) */
    short           pr_nsysarg;        /* Number of arguments to this */
                                       /* syscall */
    long            pr_sysarg[PRSYSARGS]; /* Arguments to this syscall */
    pid_t           pr_pid;            /* Process id */
```

```
    pid_t       pr_ppid;        /* Parent process id */
    pid_t       pr_pgrp;        /* Process group id */
    pid_t       pr_sid;         /* Session id */
    timestruc_t pr_utime;       /* Process user cpu time */
    timestruc_t pr_stime;       /* Process system cpu time */
    timestruc_t pr_cutime;      /* Sum of children's user times */
    timestruc_t pr_cstime;      /* Sum of children's system times */
    char        pr_clname[PRCLSZ]; /* Scheduling class name */
    long        pr_instr;       /* Current instruction */
    prgregset_t pr_reg;         /* General registers */
} prstatus_t;
```

pr_flags is a bit-mask holding these flags:

PR_STOPPED	*lwp* is stopped
PR_ISTOP	*lwp* is stopped on an event of interest (see **PIOCSTOP**)
PR_DSTOP	*lwp* has a stop directive in effect (see **PIOCSTOP**)
PR_STEP	*lwp* has a single-step directive in effect (see **PIOCRUN**)
PR_ASLEEP	*lwp* is in an interruptible sleep within a system call
PR_PCINVAL	*lwp*'s current instruction (**pr_instr**) is undefined
PR_ISSYS	process is a system process (see **PIOCSTOP**)
PR_PTRACE	process is controlled by **ptrace()** (obsolete and never set)
PR_FORK	process has its inherit-on-fork flag set (see **PIOCSET**)
PR_RLC	process has its run-on-last-close flag set (see **PIOCSET**)
PR_KLC	process has its kill-on-last-close flag set (see **PIOCSET**)
PR_ASYNC	process has its asynchronous-stop flag set (see **PIOCSET**)
PR_PCOMPAT	process has its ptrace-compatibility flag set (see **PIOCSET**)

pr_why and **pr_what** together describe, for a stopped *lwp*, the reason for the stop. Possible values of **pr_why** are:

PR_REQUESTED	indicates that the stop occurred in response to a stop directive, normally because **PIOCSTOP** was applied or because another *lwp* stopped on an event of interest and the asynchronous-stop flag (see **PIOCSET**) was not set for the process. **pr_what** is unused in this case.
PR_SIGNALLED	indicates that the *lwp* stopped on receipt of a signal (see **PIOCSTRACE**); **pr_what** holds the signal number that caused the stop (for a newly-stopped *lwp*, the same value is in **pr_cursig**).
PR_FAULTED	indicates that the *lwp* stopped on incurring a hardware fault (see **PIOCSFAULT**); **pr_what** holds the fault number that caused the stop.
PR_SYSENTRY and **PR_SYSEXIT**	indicate a stop on entry to or exit from a system call (see **PIOCSENTRY** and **PIOCSEXIT**); **pr_what** holds the system call number.

PR_JOBCONTROL indicates that the *lwp* stopped due to the default action of a job control stop signal (see **sigaction**(2)); **pr_what** holds the stopping signal number.

PR_SUSPENDED indicates that the *lwp* stopped due to internal synchronization of *lwp*s within the process. *lwp*. **pr_what** is unused in this case.

pr_who names the specific *lwp*. **pr_nlwp** is the total number of *lwp*s in the process.

pr_cursig names the current signal, that is, the next signal to be delivered to the *lwp*. **pr_sigpend** identifies any other signals pending for the process. **pr_lwppend** identifies any synchronously-generated or directed signals pending for the *lwp*. **pr_sighold** identifies those signals whose delivery is being delayed if sent to the *lwp*.

pr_info, when the *lwp* is in a PR_SIGNALLED or PR_FAULTED stop, contains additional information pertinent to the particular signal or fault (see **<sys/siginfo.h>**).

pr_altstack contains the alternate signal stack information for the *lwp* (see **sigaltstack**(2)). **pr_action** contains the signal action information pertaining to the current signal (see **sigaction**(2)); it is undefined if **pr_cursig** is zero.

pr_oldcontext, if not **NULL**, contains the address in the process of a **ucontext** structure describing the previous user-level context (see **ucontext**(5)). It is non-**NULL** only if the *lwp* is executing in the context of a signal handler and is the same as the **ucontext** pointer passed to the signal handler.

pr_brkbase is the virtual address of the process heap and **pr_brksize** is its size in bytes. The address formed by the sum of these values is the process *break* (see **brk**(2)). **pr_stkbase** and **pr_stksize** are, respectively, the virtual address of the process stack and its size in bytes. (Each *lwp* runs on a separate stack; the distinguishing characteristic of the ''process stack'' is that the operating system will grow it when necessary.)

pr_syscall is the number of the system call, if any, being executed by the *lwp*; it is non-zero only if the *lwp* is stopped on PR_SYSENTRY or PR_SYSEXIT or is asleep within a system call (PR_ASLEEP is set). If **pr_syscall** is non-zero, **pr_nsysarg** is the number of arguments to the system call and the **pr_sysarg** array contains the actual arguments.

pr_pid, **pr_ppid**, **pr_pgrp**, and **pr_sid** are, respectively, the process id, the id of the process's parent, the process's process group id, and the process's session id.

pr_utime, **pr_stime**, **pr_cutime**, and **pr_cstime** are, respectively, the user CPU and system CPU time consumed by the process, and the cumulative user CPU and system CPU time consumed by the process's children, in seconds and nanoseconds.

pr_clname contains the name of the *lwp*'s scheduling class.

SPARC: **pr_instr** contains the machine instruction to which the *lwp*'s program counter refers. The amount of data retrieved from the process is machine-dependent; on SPARC machines, it is a 32-bit word. In general, the size is that of the machine's smallest instruction. If PR_PCINVAL is set, **pr_instr** is undefined; this occurs whenever the *lwp* is not stopped or when the program counter refers to an invalid virtual address.

SPARC: **pr_reg** is an array holding the contents of a stopped *lwp*'s general registers. On SPARC machines the predefined constants **R_G0** ... **R_G7**, **R_O0** ... **R_O7**, **R_L0** ... **R_L7**, **R_I0** ... **R_I7**, **R_PSR**, **R_PC**, **R_nPC**, **R_Y**, **R_WIM**, and **R_TBR** can be used as indices to refer to the corresponding registers; previous register windows can be read from their overflow locations on the stack (see, however, **PIOCGWIN**). If the *lwp* is not stopped, all register values are undefined.

x86: **pr_instr** contains the machine instruction to which the lwp's program counter refers. The amount of data retrieved from the process is machine-dependent; on x86 machines, it is a 32-bit word. In general, the size is that of the machine's smallest instruction. If PR_PCINVAL is set, **pr_instr** is undefined; this occurs whenever the lwp is not stopped or when the program counter refers to an invalid virtual address.

x86: **pr_reg** is an array holding the contents of a stopped *lwp*'s general registers. On x86 machines, the predefined constants **SS**, **UESP**, **EFL**, **CS**, **EIP**, **ERR**, **TRAPNO**, **EAX**, **ECX**, **EDX**, **EBX**, **ESP**, **EBP**, **ESI**, **EDI**, **DS**, **ES**, **FS**, and **GS** can be used as indices to refer to the corresponding registers. If the *lwp* is not stopped, all register values are undefined.

When applied to an *lwp* file descriptor, **PIOCSTATUS** returns the status for the specific *lwp*. When applied to the process file descriptor, an *lwp* is chosen by the system for the operation. The chosen *lwp* is a stopped *lwp* only if all of the process's *lwps* are stopped, is stopped on an event of interest only if all of the *lwp*s are so stopped (excluding PR_SUSPENDED *lwps*), is in a PR_REQUESTED stop only if there are no other events of interest to be found, or failing everything else is in a PR_SUSPENDED stop (implying that the process is deadlocked). The chosen *lwp* remains fixed so long as all of the *lwp*s are either stopped on events of interest or are PR_SUSPENDED and **PIOCRUN** is not applied to any of them.

When applied to the process file descriptor, every **/proc ioctl** operation that must act on an *lwp* uses the same algorithm to choose which *lwp* to act upon. Together with synchronous stopping (see **PIOCSET**), this enables a debugger to control a multiple- *lwp* process using only the process file descriptor if it so chooses. More fine-grained control can be achieved using individual *lwp* file descriptors.

PIOCLSTATUS The **PIOCLSTATUS** operation fills in an array of **prstatus** structures addressed by *p*, one element (one structure) for each *lwp* in the process, containing the status that would be returned by applying **PIOCSTATUS** to the corresponding *lwp* file descriptor, plus an additional element at the beginning containing the status that would be returned by applying **PIOCSTATUS** to the process file descriptor.

*__PIOCSTOP__
__PIOCWSTOP__ When applied to the process file descriptor, **PIOCSTOP** directs all *lwp*s to stop and waits for them to stop; **PIOCWSTOP** simply waits for all *lwp*s to stop. When applied to an *lwp* file descriptor, **PIOCSTOP** directs the specific *lwp* to stop and waits until it has stopped; **PIOCWSTOP** simply waits for the *lwp* to stop. When applied to an *lwp* file descriptor, these operations complete when the *lwp* stops on an event of interest, immediately if already so stopped. When applied to the process file descriptor, they complete when every *lwp* has stopped on an event of interest or has come to a PR_SUSPENDED stop. If *p* is non-zero it points to a **prstatus** structure to be filled with status information for the

specific or chosen stopped *lwp* (see **PIOCSTATUS**).

An ''event of interest'' is either a PR_REQUESTED stop or a stop that has been specified in the process's tracing flags (set by **PIOCSTRACE**, **PIOCSFAULT**, **PIOCSENTRY**, and **PIOCSEXIT**). PR_JOBCONTROL and PR_SUSPENDED stops are specifically not events of interest. (An *lwp* may stop twice due to a stop signal, first showing PR_SIGNALLED if the signal is traced and again showing PR_JOBCONTROL if the *lwp* is set running without clearing the signal.) If **PIOCSTOP** is applied to an *lwp* that is stopped, but not on an event of interest, the stop directive takes effect when the *lwp* is restarted by the competing mechanism; at that time the *lwp* enters a PR_REQUESTED stop before executing any user-level code.

ioctls are interruptible by signals so that, for example, an **alarm**(2) can be set to avoid waiting forever for a process or *lwp* that may never stop on an event of interest. If **PIOCSTOP** is interrupted, the *lwp* stop directives remain in effect even though the **ioctl** returns an error.

A system process (indicated by the PR_ISSYS flag) never executes at user level, has no user-level address space visible through **/proc**, and cannot be stopped. Applying **PIOCSTOP** or **PIOCWSTOP** to a system process or any of its *lwp*s elicits the error EBUSY.

∗**PIOCRUN** An *lwp* is made runnable again after a stop. If *p* is non-zero it points to a **prrun** structure describing additional actions to be performed. The **prrun** structure contains at least the following fields:

```
typedef struct prrun {
        long        pr_flags;     /* Flags */
        sigset_t    pr_trace;     /* Set of signals to be traced */
        sigset_t    pr_sighold;   /* Set of signals to be held */
        fltset_t    pr_fault;     /* Set of faults to be traced */
        caddr_t     pr_vaddr;     /* Virtual address at which to resume */
} prrun_t;
```

pr_flags is a bit-mask describing optional actions; the remainder of the entries are meaningful only if the appropriate bits are set in **pr_flags**. Flag definitions:

PRCSIG	clears the current signal, if any (see **PIOCSSIG**).
PRCFAULT	clears the current fault, if any (see **PIOCCFAULT**).
PRSTRACE	sets the traced signal set to **pr_trace** (see **PIOCSTRACE**).
PRSHOLD	sets the held signal set to **pr_sighold** (see **PIOCSHOLD**).
PRSFAULT	sets the traced fault set to **pr_fault** (see **PIOCSFAULT**).
PRSVADDR	sets the address at which execution resumes to **pr_vaddr**.
PRSTEP	directs the *lwp* to execute a single machine instruction. On completion of the instruction, a trace trap occurs. If FLTTRACE is being traced, the *lwp* stops, otherwise it is sent **SIGTRAP**; if **SIGTRAP** is being traced and not held, the *lwp* stops. When the *lwp* stops on an event of interest the single-step directive is cancelled, even if the stop occurs before the instruction is executed.

This operation requires hardware and operating system support and may not be implemented on all processors. It is implemented on SPARC and x86 machines.

PRSABORT is meaningful only if the *lwp* is in a PR_SYSENTRY stop or is marked PR_ASLEEP; it instructs the *lwp* to abort execution of the system call (see **PIOCSENTRY**, **PIOCSEXIT**).

PRSTOP directs the *lwp* to stop again as soon as possible after resuming execution (see **PIOCSTOP**). In particular if the *lwp* is stopped on PR_SIGNALLED or PR_FAULTED, the next stop will show PR_REQUESTED, no other stop will have intervened, and the *lwp* will not have executed any user-level code.

When applied to an *lwp* file descriptor **PIOCRUN** makes the specific *lwp* runnable. The operation fails (**EBUSY**) if the specific *lwp* is not stopped on an event of interest.

When applied to the process file descriptor an *lwp* is chosen for the operation as described under **PIOCSTATUS**. The operation fails (**EBUSY**) if the chosen *lwp* is not stopped on an event of interest. If PRSTEP or PRSTOP was requested, the chosen *lwp* is made runnable; otherwise, the chosen *lwp* is marked PR_REQUESTED. If as a consequence all *lwp*s are in the PR_REQUESTED or PR_SUSPENDED stop state, all *lwp*s showing PR_REQUESTED are made runnable.

PIOCLWPIDS This returns, in an array of **id_t**'s addressed by *p*, the *lwp* identifiers of all the *lwp*s that exist in the process, plus an extra identifier containing zero to mark the end of the list. The number of *lwp*s in the process can be determined from the **pr_nlwp** field of the **prstatus** structure.

PIOCNLDT
PIOCLDT These operations apply only to x86 machines. They provide read-only access to the traced process's local descriptor table (LDT). A process's LDT is maintained by the operating system. **PIOCNLDT** returns, in an integer addressed by *p*, the number of LDT entries currently active. This value can be used with the **PIOCLDT** operation. The **PIOCLDT** operation returns the set of currently active LDT entries. For **PIOCLDT**, *p* addresses an array of elements of type **struct ssd**, defined in **<sys/sysi86.h>**. One array element (one structure) is returned for each active LDT entry, plus an additional element containing all zeroes to mark the end of the list.

PIOCOPENLWP The return value *retval* provides a **/proc** file descriptor that refers to the *lwp* named in the **id_t** addressed by *p*. The read/write attributes of the newly-acquired file descriptor are the same as those of the file descriptor used in the operation. The new file descriptor has an independent file offset for **lseek**(2). On error (no such *lwp*), –1 is returned and **errno** is set to **ENOENT**.

∗**PIOCSTRACE** This defines a set of signals to be traced in the process: the receipt of one of these signals by an *lwp* causes the *lwp* to stop. The set of signals is defined via an instance of **sigset_t** addressed by *p*. Receipt of **SIGKILL** cannot be traced; if specified, it is silently ignored.

If a signal that is included in an *lwp*'s held signal set is sent to the *lwp*, the signal is not received and does not cause a stop until it is removed from the held signal set, either by the *lwp* itself or by setting the held signal set with **PIOCSHOLD** or the PRSHOLD option of **PIOCRUN**.

PIOCGTRACE The current traced signal set is returned in an instance of **sigset_t** addressed by *p*.

∗**PIOCSSIG** The current signal and its associated signal information for the specific or chosen *lwp* are set according to the contents of the **siginfo** structure addressed by *p* (see **<sys/siginfo.h>**). If the specified signal number is zero or if *p* is zero, the current signal is cleared. The semantics of this operation are different from that of **kill**(2) or **PIOCKILL** in that the signal is delivered to the *lwp* immediately after execution is resumed (even if the signal is being held) and an additional PR_SIGNALLED stop does not intervene even if the signal is being traced. Setting the current signal to **SIGKILL** terminates the process immediately.

∗**PIOCKILL** If applied to the process file descriptor, a signal is sent to the process with semantics identical to that of **kill**(2). If applied to an *lwp* file descriptor, a signal is sent to the specific *lwp*. *lwp* with semantics identical to that of *p* points to an *int* naming the signal. Sending **SIGKILL** terminates the process immediately, even if the signal is sent to a specific *lwp*.

∗**PIOCUNKILL** A signal is deleted, that is, it is removed from the set of pending signals. If applied to the process file descriptor, the signal is deleted from the process's pending signals. If applied to an *lwp* file descriptor, the signal is deleted from the *lwp*'s pending signals. The current signal (if any) is unaffected. *p* points to an *int* naming the signal. It is an error (**EINVAL**) to attempt to delete **SIGKILL**.

PIOCGHOLD **PIOCGHOLD** returns the set of held signals for the specific or chosen *lwp* (signals whose
∗**PIOCSHOLD** delivery will be delayed if sent to the *lwp*) in an instance of **sigset_t** addressed by *p*. **PIOCSHOLD** correspondingly sets the *lwp*'s held signal set but does not allow **SIGKILL** or **SIGSTOP** to be held; if specified, they are silently ignored.

PIOCMAXSIG These operations provide information about the signal actions associated with the traced
PIOCACTION process (see **sigaction**(2)). **PIOCMAXSIG** returns, in the *int* addressed by *p*, the maximum signal number understood by the system. This can be used to allocate storage for use with the **PIOCACTION** operation, which returns the traced process's signal actions in an array of **sigaction** structures addressed by *p*. Signal numbers are displaced by 1 from array indices, so that the action for signal number *n* appears in position *n*-1 of the array.

∗**PIOCSFAULT** This defines a set of hardware faults to be traced in the process: on incurring one of these faults an *lwp* stops. The set is defined via an instance of **fltset_t** addressed by *p*. Fault names are defined in **<sys/fault.h>** and include the following. Some of these may not occur on all processors; there may be processor-specific faults in addition to these.

FLTILL	illegal instruction
FLTPRIV	privileged instruction
FLTBPT	breakpoint trap
FLTTRACE	trace trap
FLTACCESS	memory access fault (bus error)
FLTBOUNDS	memory bounds violation
FLTIOVF	integer overflow
FLTIZDIV	integer zero divide
FLTFPE	floating-point exception
FLTSTACK	unrecoverable stack fault
FLTPAGE	recoverable page fault

When not traced, a fault normally results in the posting of a signal to the *lwp* that incurred the fault. If an *lwp* stops on a fault, the signal is posted to the *lwp* when execution is resumed unless the fault is cleared by **PIOCCFAULT** or by the PRCFAULT option of **PIOCRUN**. FLTPAGE is an exception; no signal is posted. There may be additional processor-specific faults like this. **pr_info** in the **prstatus** structure identifies the signal to be sent and contains machine-specific information about the fault.

PIOCGFAULT The current traced fault set is returned in an instance of **fltset_t** addressed by *p*.

∗**PIOCCFAULT** The current fault (if any) is cleared; the associated signal is not sent to the specific or chosen *lwp*.

∗**PIOCSENTRY**
∗**PIOCSEXIT**

These operations instruct the process's *lwp*s to stop on entry to or exit from specified system calls. The set of system calls to be traced is defined via an instance of **sysset_t** addressed by *p*.

When entry to a system call is being traced, an *lwp* stops after having begun the call to the system but before the system call arguments have been fetched from the *lwp*. When exit from a system call is being traced, an *lwp* stops on completion of the system call just prior to checking for signals and returning to user level. At this point all return values have been stored into the *lwp*'s registers.

If an *lwp* is stopped on entry to a system call (PR_SYSENTRY) or when sleeping in an interruptible system call (PR_ASLEEP is set), it may be instructed to go directly to system call exit by specifying the PRSABORT flag in a **PIOCRUN** request. Unless exit from the system call is being traced the *lwp* returns to user level showing error EINTR.

PIOCGENTRY
PIOCGEXIT

These return the current traced system call entry or exit set in an instance of **sysset_t** addressed by *p*.

∗**PIOCSET**
∗**PIOCRESET**

PIOCSET sets one or more modes of operation for the traced process. **PIOCRESET** resets these modes. The modes to be set or reset are specified by flags in a *long* addressed by *p*:

PR_FORK (inherit-on-fork): When set, the process's tracing flags are inherited by the child of a **fork**(2) or **vfork**(2). When reset, child processes start with all tracing flags cleared.

PR_RLC (run-on-last-close): When set and the last writable **/proc** file descriptor referring to the traced process or any of its *lwp*s is closed, all of the process's tracing flags are cleared, any outstanding stop directives are canceled, and if any *lwp*s are stopped on events of interest, they are set running as though **PIOCRUN** had been applied to them. When reset, the process's tracing flags are retained and *lwp*s are not set running on last close.

PR_KLC (kill-on-last-close): When set and the last writable **/proc** file descriptor referring to the traced process or any of its *lwp*s is closed, the process is terminated with **SIGKILL**.

PR_ASYNC (asynchronous-stop): When set, a stop on an event of interest by one *lwp* does not directly affect any other *lwp* in the process. When reset and an *lwp* stops on an event of interest other than PR_REQUESTED, all other *lwp*s in the process are directed to stop.

PR_PCOMPAT (ptrace-compatibility): When set, a stop on an event of interest by the traced process is reported to the parent of the traced process via **wait**(2), **SIGTRAP** is sent to the traced process when it executes a successful **exec**(2), setuid/setgid flags are not honored for execs performed by the traced process, any exec of an object file that the traced process cannot read fails, and the traced process dies when its parent dies. This mode is deprecated; it is provided only to allow **ptrace()** to be implemented as a library function using **/proc**.

It is an error (**EINVAL**) to specify flags other than those described above or to apply these operations to a system process. The current modes are reported in the **prstatus** structure (see **PIOCSTATUS**).

∗**PIOCSFORK**
∗**PIOCRFORK**

PIOCSFORK sets the inherit-on-fork flag in the traced process. **PIOCRFORK** turns this flag off. (Obsolete, see **PIOCSET**.)

∗**PIOCSRLC**
∗**PIOCRRLC**

PIOCSRLC sets the run-on-last-close flag in the traced process. **PIOCRRLC** turns this flag off. (Obsolete, see **PIOCSET**.)

PIOCGREG
∗**PIOCSREG**

These operations respectively get and set the general registers for the specific or chosen *lwp* into or out of an array addressed by *p*; the array has type **prgregset_t**. Register contents are accessible using a set of predefined indices (see **PIOCSTATUS**).

On SPARC systems, only certain bits of the processor-status register (R_PS) can be modified by **PIOCSREG**: these include only the condition-code bits. Other privileged registers cannot be modified at all.

On x86 systems, only certain bits of the flags register (EFL) can be modified by **PIOCSREG**: these include the condition codes, direction-bit, trace-bit, and overflow-bit.

PIOCSREG fails (**EBUSY**) if the *lwp* is not stopped on an event of interest. If the *lwp* is not stopped, the register values returned by **PIOCGREG** are undefined.

PIOCGFPREG
∗**PIOCSFPREG**

These operations respectively get and set the floating-point registers for the specific or chosen *lwp* into or out of a structure addressed by *p*; the structure has type **prfpregset_t**. An error (**EINVAL**) is returned if the system does not support floating-point operations

(no floating-point hardware and the system does not emulate floating-point machine instructions). **PIOCSFPREG** fails (**EBUSY**) if the *lwp* is not stopped on an event of interest. If the *lwp* is not stopped, the register values returned by **PIOCGFPREG** are undefined.

***PIOCNICE** The traced process's **nice**(2) priority is incremented by the amount contained in the *int* addressed by *p*. Only the super-user may better a process's priority in this way, but any user may lower the priority. This operation is meaningful only when applied to a process in the time-sharing scheduling class.

PIOCPSINFO This returns miscellaneous process information such as that reported by **ps**(1). *p* is a pointer to a **prpsinfo** structure containing at least the following fields:

```
typedef struct prpsinfo {
        char        pr_state;       /* numeric process state (see pr_sname) */
        char        pr_sname;       /* printable character representing pr_state */
        char        pr_zomb;        /* !=0: process terminated but not waited for */
        char        pr_nice;        /* nice for cpu usage */
        u_long      pr_flag;        /* process flags */
        uid_t       pr_uid;         /* real user id */
        gid_t       pr_gid;         /* real group id */
        pid_t       pr_pid;         /* unique process id */
        pid_t       pr_ppid;        /* process id of parent */
        pid_t       pr_pgrp;        /* pid of process group leader */
        pid_t       pr_sid;         /* session id */
        caddr_t     pr_addr;        /* physical address of process */
        long        pr_size;        /* size of process image in pages */
        long        pr_rssize;      /* resident set size in pages */
        u_long      pr_bysize;      /* size of process image in bytes */
        u_long      pr_byrssize;    /* resident set size in bytes */
        caddr_t     pr_wchan;       /* wait addr for sleeping process */
        short       pr_syscall;     /* system call number (if in syscall) */
        timestruc_t pr_start;       /* process start time, sec+nsec since epoch */
        timestruc_t pr_time;        /* usr+sys cpu time for this process */
        timestruc_t pr_ctime;       /* usr+sys cpu time for reaped children */
        long        pr_pri;         /* priority, high value is high priority */
        char        pr_oldpri;      /* pre-SVR4, low value is high priority */
        char        pr_cpu;         /* pre-SVR4, cpu usage for scheduling */
        dev_t       pr_ttydev;      /* controlling tty device (PRNODEV if none) */
        char        pr_clname[PRCLSZ];  /* scheduling class name */
        char        pr_fname[PRFNSZ];   /* last component of exec()ed pathname */
        char        pr_psargs[PRARGSZ]; /* initial characters of arg list */
} prpsinfo_t;
```

Some of the entries in **prpsinfo**, such as **pr_state** and **pr_flag**, are system-specific and should not be expected to retain their meanings across different versions of the operating system. **pr_addr** is a vestige of the past and has no real meaning in current systems.

PIOCPSINFO can be applied to a *zombie* process (one that has terminated but whose parent has not yet performed a **wait**(2) on it).

PIOCNMAP
PIOCMAP

These operations provide information about the memory mappings (virtual address ranges) associated with the traced process. **PIOCNMAP** returns, in the *int* addressed by *p*, the number of mappings that are currently active. This can be used to allocate storage for use with the **PIOCMAP** operation, which returns the list of currently active mappings. For **PIOCMAP**, *p* addresses an array of elements of type **prmap_t**; one array element (one structure) is returned for each mapping, plus an additional element containing all zeros to mark the end of the list. The **prmap** structure contains at least the following fields:

```
typedef struct prmap {
        caddr_t   pr_vaddr;     /* Virtual address */
        u_long    pr_size;      /* Size of mapping in bytes */
        u_long    pr_pagesize;  /* pagesize in bytes for this mapping */
        off_t     pr_off;       /* Offset into mapped object, if any */
        long      pr_mflags;    /* Protection and attribute flags */
} prmap_t;
```

pr_vaddr is the virtual address of the mapping within the traced process and **pr_size** is its size in bytes. **pr_pagesize** is the size in bytes of virtual memory pages for this mapping. **pr_off** is the offset within the mapped object (if any) to which the virtual address is mapped.

pr_mflags is a bit-mask of protection and attribute flags:

MA_READ	mapping is readable by the traced process
MA_WRITE	mapping is writable by the traced process
MA_EXEC	mapping is executable by the traced process
MA_SHARED	mapping changes are shared by the mapped object
MA_BREAK	mapping is grown by the **brk**(2) system call (obsolete)
MA_STACK	mapping is grown automatically on stack faults (obsolete)

A contiguous area of the address space having the same underlying mapped object may appear as multiple mappings due to varying read/write/execute/shared attributes. The underlying mapped object does not change over the range of a single mapping. An I/O operation to a mapping marked MA_SHARED fails if applied at a virtual address not corresponding to a valid page in the underlying mapped object. A write to a MA_SHARED mapping that is not marked MA_WRITE fails. Reads and writes to private mappings always succeed. Reads and writes to unmapped addresses always fail.

The MA_BREAK and MA_STACK flags are provided only for compatibility with older versions of the system and should not be relied upon. The **pr_brkbase**, **pr_brksize**, **pr_stkbase** and **pr_stksize** members of the **prstatus** structure should be used instead.

PIOCOPENM

The return value *retval* provides a read-only file descriptor for a mapped object associated with the traced process. If *p* is zero the traced process's executable file is found. This enables a debugger to find the object file symbol table without having to know the path name of the executable file. If *p* is non-zero it points to a **caddr_t** containing a virtual address within the traced process and the mapped object, if any, associated with that

address is found; this can be used to get a file descriptor for a shared library that is attached to the process. On error (invalid address or no mapped object for the designated address), –1 is returned and **errno** is set to **EINVAL**.

PIOCCRED Fetch the set of credentials associated with the process. *p* points to an instance of **prcred_t** which is filled by the operation. The **prcred** structure contains at least the following fields:

```
typedef struct prcred {
        uid_t   pr_euid;        /* Effective user id */
        uid_t   pr_ruid;        /* Real user id */
        uid_t   pr_suid;        /* Saved user id (from exec) */
        gid_t   pr_egid;        /* Effective group id */
        gid_t   pr_rgid;        /* Real group id */
        gid_t   pr_sgid;        /* Saved group id (from exec) */
        u_int   pr_ngroups;     /* Number of supplementary groups */
} prcred_t;
```

PIOCGROUPS Fetch the set of supplementary group IDs associated with the process. *p* points to an array of elements of type **gid_t**, which will be filled by the operation. **PIOCCRED** can be applied beforehand to determine the number of groups (**pr_ngroups**) that will be returned and the amount of storage that should be allocated to hold them.

PIOCNAUXV **PIOCAUXV** These operations provide values of entries in the aux vector that is passed by the operating system as startup information to the dynamic loader. **PIOCNAUXV** returns, in the *int* addressed by *p*, the number of available aux vector entries. This can be used to allocate storage for use with the **PIOCAUXV** operation, which returns the initial values of the process's aux vector in an array of **auxv_t** structures addressed by *p* (see **<sys/auxv.h>**).

PIOCUSAGE When applied to the process file descriptor, **PIOCUSAGE** returns the process usage information; when applied to an *lwp* file descriptor, it returns usage information for the specific *lwp*. *p* points to a **prusage** structure which is filled by the operation. The **prusage** structure contains at least the following fields:

```
typedef struct prusage {
        id_t            pr_lwpid;       /* lwp id.  0: process or defunct */
        u_long          pr_count;       /* number of contributing lwps */
        timestruc_t     pr_tstamp;      /* current time stamp */
        timestruc_t     pr_create;      /* process/lwp creation time stamp */
        timestruc_t     pr_term;        /* process/lwp termination time stamp */
        timestruc_t     pr_rtime;       /* total lwp real (elapsed) time */
        timestruc_t     pr_utime;       /* user level CPU time */
        timestruc_t     pr_stime;       /* system call CPU time */
        timestruc_t     pr_ttime;       /* other system trap CPU time */
        timestruc_t     pr_tftime;      /* text page fault sleep time */
        timestruc_t     pr_dftime;      /* data page fault sleep time */
        timestruc_t     pr_kftime;      /* kernel page fault sleep time */
```

```
        timestruc_t     pr_ltime;       /* user lock wait sleep time */
        timestruc_t     pr_slptime;     /* all other sleep time */
        timestruc_t     pr_wtime;       /* wait-cpu (latency) time */
        timestruc_t     pr_stoptime;    /* stopped time */
        u_long          pr_minf;        /* minor page faults */
        u_long          pr_majf;        /* major page faults */
        u_long          pr_nswap;       /* swaps */
        u_long          pr_inblk;       /* input blocks */
        u_long          pr_oublk;       /* output blocks */
        u_long          pr_msnd;        /* messages sent */
        u_long          pr_mrcv;        /* messages received */
        u_long          pr_sigs;        /* signals received */
        u_long          pr_vctx;        /* voluntary context switches */
        u_long          pr_ictx;        /* involuntary context switches */
        u_long          pr_sysc;        /* system calls */
        u_long          pr_ioch;        /* chars read and written */
} prusage_t;
```

PIOCUSAGE can be applied to a *zombie* process (see **PIOCPSINFO**).

PIOCLUSAGE The **PIOCLUSAGE** operation fills in an array of **prusage** structures addressed by *p*, one element for each *lwp* in the process plus an additional element at the beginning that contains the summation over all defunct *lwp*s (*lwp*s that once existed but no longer exist in the process). Excluding the **pr_lwpid**, **pr_tstamp**, **pr_create** and **pr_term** entries, the entry-by-entry summation over all these structures is the definition of the process usage information.

PIOCLUSAGE can be applied to a *zombie* process (see **PIOCPSINFO**).

PIOCOPENPD The return value *retval* provides a read-only file descriptor for a ''page data file'', enabling tracking of address space references and modifications on a per-page basis.

A **read**(2) of the page data file descriptor returns structured page data and atomically clears the page data maintained for the file by the system. That is to say, each read returns data collected since the last read; the first read returns data collected since the file was opened. When the call completes, the read buffer contains the following structure as its header and thereafter contains a number of section header structures and associated byte arrays that must be accessed by walking linearly through the buffer.

```
typedef struct prpageheader {
        timestruc_t     pr_tstamp;      /* real time stamp */
        u_long          pr_nmap;        /* number of address space mappings */
        u_long          pr_npage;       /* total number of pages */
} prpageheader_t;
```

The header is followed by **pr_nmap prasmap** structures and associated data arrays. The **prasmap** structure contains at least the following elements.

```
typedef struct prasmap {
        caddr_t         pr_vaddr;       /* virtual address */
        u_long          pr_npage;       /* number of pages in mapping */
        off_t           pr_off;         /* offset into mapped object, if any */
        u_long          pr_mflags;      /* protection and attribute flags */
        u_long          pr_pagesize;    /* pagesize in bytes for this mapping */
} prasmap_t;
```

Each section header is followed by **pr_npage** bytes, one byte for each page in the mapping, plus enough null bytes at the end so that the next **prasmap** structure begins on a long-aligned boundary. Each data byte may contain these flags:

PG_REFERENCED	page has been referenced
PG_MODIFIED	page has been modified

If the read buffer is not large enough to contain all of the page data, the read fails with E2BIG and the page data is not cleared. The required size of the read buffer can be determined through **fstat**(2). Application of **lseek**(2) to the page data file descriptor is ineffective. Closing the page data file descriptor terminates the system overhead associated with collecting the data.

More than one page data file descriptor for the same process can be opened, up to a system-imposed limit per traced process. A read of one does not affect the data being collected by the system for the others.

The **PIOCOPENPD** operation returns -1 on failure. Reasons for failure are application to a system process (**EINVAL**) or too many page data file descriptors were requested (**ENOMEM**).

PIOCGWIN This operation applies only to SPARC architecture machines. *p* is a pointer to a **gwindows_t** structure, defined in **<sys/reg.h>**, that is filled with the contents of those SPARC register windows that could not be stored on the stack when the *lwp* stopped. Conditions under which register windows are not stored on the stack are: the stack pointer refers to nonexistent process memory or the stack pointer is improperly aligned. If the specific or chosen *lwp* is not stopped, the operation returns undefined values.

PIOCGETPR **PIOCGETU** These operations copy, respectively, the traced process's *proc* structure and *user* structure into the buffer addressed by *p*. They are provided for completeness but it should be unnecessary to access either of these structures directly since relevant status information is available through other control operations. Their use is discouraged because a program making use of them is tied to a particular version of the operating system.

PIOCGETPR can be applied to a *zombie* process (see **PIOCPSINFO**).

FILES

/proc	directory (list of processes)
/proc/*nnnnn*	process file

SEE ALSO **alarm**(2), **brk**(2), **close**(2), **exec**(2), **fork**(2), **ioctl**(2), **kill**(2), **lseek**(2), **nice**(2), **open**(2), **poll**(2), **read**(2), **sigaction**(2), **wait**(2), **signal**(3C), **siginfo**(5), **signal**(5),

DIAGNOSTICS

Errors that can occur in addition to the errors normally associated with file system access:

ENOENT The traced process or *lwp* has terminated after being opened.

EIO I/O was attempted at an illegal address in the traced process.

EBADF An I/O or **ioctl** operation requiring write access was attempted on a file descriptor not open for writing.

EBUSY **PIOCSTOP** or **PIOCWSTOP** was applied to a system process; an exclusive **open**(2) was attempted on a process file already already open for writing; an **open**(2) for writing was attempted and an exclusive open is in effect on the process file; **PIOCRUN**, **PIOCSREG** or **PIOCSFPREG** was applied to a process or *lwp* not stopped on an event of interest; an attempt was made to mount **/proc** when it is already mounted.

EPERM Someone other than the super-user attempted to better a process's priority by issuing **PIOCNICE**.

ENOSYS An attempt was made to perform an unsupported operation (such as create, remove, link, or unlink) on an entry in **/proc**.

EFAULT An I/O or **ioctl** request referred to an invalid address in the controlling process.

EINVAL In general this means that some invalid argument was supplied to a system call. The list of conditions eliciting this error includes: the **ioctl** code is undefined; an **ioctl** operation was issued on a file descriptor referring to the **/proc** directory; the PRSTEP option of the **PIOCRUN** operation was used on an implementation that does not support single-stepping; an out-of-range signal number was specified with **PIOCSSIG**, **PIOCKILL**, or **PIOCUNKILL**; **SIGKILL** was specified with **PIOCUNKILL**; an illegal virtual address was specified in a **PIOCOPENM** request; **PIOCGFPREG** or **PIOCSFPREG** was issued on a system that does not support floating-point operations.

ENOMEM The system-imposed limit on the number of page data file descriptors was reached on a **PIOCOPENPD** request.

E2BIG Data to be returned in a **read**(2) of the page data file exceeds the size of the read buffer provided by the caller.

EINTR A signal was received by the controlling process while waiting for the traced process or *lwp* to stop via **PIOCSTOP** or **PIOCWSTOP**.

EAGAIN The traced process has performed an **exec**(2) of a setuid/setgid object file or of an object file that it cannot read; all further operations on the process or *lwp* file descriptor (except **close**(2)) elicit this error.

NOTES

Each operation (**ioctl** or I/O) is guaranteed to be atomic with respect to the traced process, except when applied to a system process. I/O to memory that is shared by another process or is the target of asynchronous I/O is not guaranteed to be atomic.

For security reasons, except for the super-user, an open of a **/proc** file fails unless both the user-ID and group-ID of the caller match those of the traced process and the process's object file is readable by the caller. Files corresponding to setuid/setgid processes can be opened only by the super-user. Even if held by the super-user, an open process or *lwp* file descriptor becomes invalid if the traced process performs an **exec**(2) of a setuid/setgid object file or an object file that it cannot read. Any operation performed on an invalid file descriptor, except **close**(2), fails with **EAGAIN**. In this situation, if any tracing flags are set and the process or any *lwp* file descriptor is open for writing, the process will have been directed to stop and its run-on-last-close flag will have been set (see **PIOCSET**). This enables a controlling process (if it has permission) to reopen the process file to get a new valid file descriptor, close the invalid file descriptors, and proceed. Just closing the invalid file descriptors causes the traced process to resume execution with no tracing flags set. Any process not currently open for writing via **/proc** but that has left-over tracing flags from a previous open and that *exec*s a setuid/setgid or unreadable object file will not be stopped but will have all its tracing flags cleared.

To wait for one or more of a set of processes or *lwp*s to stop, **/proc** file descriptors can be used in a **poll**(2) system call. When requested and returned, the polling event POLLPRI indicates that the process or *lwp* stopped on an event of interest. Although they cannot be requested, the polling events POLLHUP, POLLERR and POLLNVAL may be returned. POLLHUP indicates that the process or *lwp* has terminated. POLLERR indicates that the file descriptor has become invalid. POLLNVAL is returned immediately if POLLPRI is requested on a file descriptor referring to a system process (see **PIOCSTOP**).

Descriptions of structures in this document include only interesting structure elements, not filler and padding fields, and may show elements out of order for descriptive clarity. The actual structure definitions are contained in **<sys/procfs.h>**.

The **PIOCLSTATUS**, **PIOCLWPIDS**, **PIOCLDT**, **PIOCMAP**, **PIOCGROUPS**, and **PIOCLUSAGE** operations return arrays whose actual sizes can only be known through previously-applied operations. Applying these operations to a process that is not stopped runs the risk of overrunning the buffer passed to the system.

For reasons of symmetry and efficiency there are more control operations than strictly necessary.

BUGS

The types **gregset_t** and **fpregset_t** defined in **<sys/reg.h>** are similar to but not the same as the types **prgregset_t** and **prfpregset_t** defined in **<sys/procfs.h>**.

NAME

profile – setting up an environment for user at login time

SYNOPSIS

/etc/profile
$HOME/.profile

DESCRIPTION

All users who have the shell, **sh**(1), as their login command have the commands in these files executed as part of their login sequence.

/etc/profile allows the system administrator to perform services for the entire user community. Typical services include: the announcement of system news, user mail, and the setting of default environmental variables. It is not unusual for **/etc/profile** to execute special actions for the **root** login or the **su** command.

The file **$HOME/.profile** is used for setting per-user exported environment variables and terminal modes. The following example is typical (except for the comments):

```
# Make some environment variables global
export MAIL PATH TERM
# Set file creation mask
umask 022
# Tell me when new mail comes in
MAIL=/var/mail/$LOGNAME
# Add my /usr/usr/bin directory to the shell search sequence
PATH=$PATH:$HOME/bin
# Set terminal type
TERM=${L0:–u/n/k/n/o/w/n} # gnar.invalid
while :
do
        if [ –f ${TERMINFO:-/usr/share/lib/terminfo}/?/$TERM ]
        then break
        elif [ –f /usr/share/lib/terminfo/?/$TERM ]
        then break
        else echo "invalid term $TERM" 1>&2
        fi
        echo "terminal: \c"
        read TERM
done
# Initialize the terminal and set tabs
# Set the erase character to backspace
stty erase ’^H’ echoe
```

FILES

$HOME/.profile	user-specific environment
/etc/profile	system-wide environment

SEE ALSO **env**(1), **login**(1), **mail**(1), **sh**(1), **stty**(1), **tput**(1), **su**(1M), **terminfo**(4), **environ**(5), **term**(5)
Solaris Advanced User's Guide

NOTES Care must be taken in providing system-wide services in **/etc/profile**. Personal **.profile** files are better for serving all but the most global needs.

NAME

protocols – protocol name database

SYNOPSIS

/etc/inet/protocols
/etc/protocols

DESCRIPTION

The **protocols** file is a local source of information regarding the known protocols used in the DARPA Internet. The protocols file can be used in conjunction with or instead of other protocols sources, including the NIS maps ‘‘protcols.byname’’ and ‘‘"protocols.bynumber’’ and the NIS+ table ‘‘protocols’’. Programs use the **getprotobyname**(3N) routine to access this information.

The protocols file has one line for each protocol. The line has the following format:

official-protocol-name protocol-number aliases

Items are separated by any number of blanks and / or TAB characters. A ‘**#**’ indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file. Protocol names may contain any printable character other than a field delimiter, NEWLINE, or comment character.

EXAMPLES

The following is a sample database:

```
#
# Internet (IP) protocols
#
ip      0       IP      # internet protocol, pseudo protocol number
icmp    1       ICMP    # internet control message protocol
ggp     3       GGP     # gateway-gateway protocol
tcp     6       TCP     # transmission control protocol
pup     12      PUP     # PARC universal packet protocol
udp     17      UDP     # user datagram protocol
```

FILES

/etc/nsswitch.conf configuration file for name-service switch

SEE ALSO

getprotobyname(3N), **nsswitch.conf**(4)

NOTES

/etc/inet/protocols is the official SVR4 name of the **protocols** file. The symbolic link **/etc/protocols** exists for BSD compatibility.

NAME

prototype – package information file

DESCRIPTION

prototype is an ASCII file used to specify package information. Each entry in the file describes a single deliverable object. An object may be a data file, directory, source file, executable object, etc. This file is generated by the package developer.

Entries in a **prototype** file consist of several fields of information separated by white space. Comment lines begin with a ''**#**'' and are ignored. The fields are described below and must appear in the order shown.

part An optional field designating the part number in which the object resides. A part is a collection of files, and is the atomic unit by which a package is processed. A developer can choose criteria for grouping files into a part (for example, based on class). If this field is not used, part 1 is assumed.

ftype A one-character field which indicates the file type. Valid values are:

- **f** a standard executable or data file
- **e** a file to be edited upon installation or removal
- **v** volatile file (one whose contents are expected to change)
- **d** directory
- **x** an exclusive directory
- **l** linked file
- **p** named pipe
- **c** character special device
- **b** block special device
- **i** installation script or information file
- **s** symbolic link

class The installation class to which the file belongs. This name must contain only alphanumeric characters and be no longer than 12 characters. The field is not specified for installation scripts. (**admin** and all classes beginning with capital letters are reserved class names.)

pathname The pathname where the file will reside on the target machine, for example, **/usr/bin/mail** or **bin/ras_proc**. Relative pathnames (those that do not begin with a slash) indicate that the file is relocatable. The form

path1=path2

may be used for two purposes: to define a link and to define local pathnames.

For linked files, *path1* indicates the destination of the link and *path2* indicates the source file. (This format is mandatory for linked files.)

For local pathnames, *path1* indicates the pathname an object should have on the machine where the entry is to be installed and *path2* indicates either a relative or fixed pathname to a file on the host machine which contains the actual contents.

A pathname may contain a variable specification, which will be resolved at the time of installation. This specification should have the form **$[A-Z]**.

major The major device number. The field is only specified for block or character special devices.

minor The minor device number. The field is only specified for block or character special devices.

mode The octal mode of the file (for example, 0664). A question mark (**?**) indicates that the mode will be left unchanged, implying that the file already exists on the target machine. This field is not used for linked files or packaging information files.

owner The owner of the file (for example, **bin** or **root**). The field is limited to 14 characters in length. A question mark (**?**) indicates that the owner will be left unchanged, implying that the file already exists on the target machine. This field is not used for linked files or packaging information files.

Can be a variable specification in the form of **$[A-Z]**. Will be resolved at installation time.

group The group to which the file belongs (for example, **bin** or **sys**). The field is limited to 14 characters in length. A question mark (**?**) indicates that the group will be left unchanged, implying that the file already exists on the target machine. This field is not used for linked files or packaging information files.

Can be a variable specification in the form of **$[A-Z]**. Will be resolved at installation time.

An exclamation point (**!**) at the beginning of a line indicates that the line contains a command. These commands are used to incorporate files in other directories, to locate objects on a host machine, and to set permanent defaults. The following commands are available:

search Specifies a list of directories (separated by white space) to search for when looking for file contents on the host machine. The basename of the *path* field is appended to each directory in the ordered list until the file is located.

include Specifies a pathname which points to another prototype file to include. Note that **search** requests do not span **include** files.

default Specifies a list of attributes (mode, owner, and group) to be used by default if attribute information is not provided for prototype entries which require the information. The defaults do not apply to entries in **include** prototype files.

param=value Places the indicated parameter in the current environment.

The above commands may have variable substitutions embedded within them, as demonstrated in the two example **prototype** files below.

Before files are overwritten during installation, they are copied to a temporary pathname. The exception to this rule is files whose mode includes execute permission, unless the file is editable (i.e, *ftype* is **e**). For files which meet this exception, the existing version is linked to a temporary pathname, and the original file is removed. This allows processes which are executing during installation to be overwritten.

EXAMPLES

Example 1:

```
!PROJDIR=/usr/proj
!BIN=$PROJDIR/bin
!CFG=$PROJDIR/cfg
!LIB=$PROJDIR/lib
!HDRS=$PROJDIR/hdrs
!search /usr/myname/usr/bin /usr/myname/src /usr/myname/hdrs
i pkginfo=/usr/myname/wrap/pkginfo
i depend=/usr/myname/wrap/depend
i version=/usr/myname/wrap/version
d none /usr/wrap 0755 root bin
d none /usr/wrap/usr/bin 0755 root bin
! search $BIN
f none /usr/wrap/bin/INSTALL 0755 root bin
f none /usr/wrap/bin/REMOVE 0755 root bin
f none /usr/wrap/bin/addpkg 0755 root bin
!default 755 root bin
f none /usr/wrap/bin/audit
f none /usr/wrap/bin/listpkg
f none /usr/wrap/bin/pkgmk
# the following file starts out zero length but grows
v none /usr/wrap/logfile=/dev/null 0644 root bin
# the following specifies a link (dest=src)
l none /usr/wrap/src/addpkg=/usr/wrap/bin/rmpkg
! search $SRC
!default 644 root other
f src /usr/wrap/src/INSTALL.sh
f src /usr/wrap/src/REMOVE.sh
f src /usr/wrap/src/addpkg.c
f src /usr/wrap/src/audit.c
f src /usr/wrap/src/listpkg.c
f src /usr/wrap/src/pkgmk.c
d none /usr/wrap/data 0755 root bin
d none /usr/wrap/save 0755 root bin
d none /usr/wrap/spool 0755 root bin
d none /usr/wrap/tmp 0755 root bin
d src /usr/wrap/src 0755 root bin
```

Example 2:

```
# this prototype is generated by 'pkgproto' to refer
# to all prototypes in my src directory
!PROJDIR=/usr/dew/projx
!include $PROJDIR/src/cmd/prototype
!include $PROJDIR/src/cmd/audmerg/protofile
!include $PROJDIR/src/lib/proto
```

SEE ALSO

pkgmk(1), **pkginfo**(4)

NOTES

Normally, if a file is defined in the **prototype** file but does not exist, that file is created at the time of package installation. However, if the file pathname includes a directory that does not exist, the file will not be created. For example, if the **prototype** file has the following entry:

```
f none /usr/dev/bin/command
```

and that file does not exist, it will be created if the directory **/usr/dev/bin** already exists or if the **prototype** also has an entry defining the directory:

```
d none /usr/dev/bin
```

NAME

pseudo – configuration files for pseudo device drivers

DESCRIPTION

Pseudo devices are devices that are implemented entirely in software. Drivers for pseudo devices must provide driver configuration files to inform the system of each pseudo device that should be created.

Configuration files for pseudo device drivers must identify the parent driver explicitly as *pseudo*, and must create an integer property called *instance* which is unique to this entry in the configuration file.

Each entry in the configuration file creates a prototype devinfo node. Each node is assigned an instance number which is determined by the value of the *instance* property. This property is only applicable to children of the *pseudo* parent, and is required since pseudo devices have no hardware address from which to determine the instance number. See **driver.conf**(4) for further details of configuration file syntax.

EXAMPLES

Here is a configuration file called **ramdisk.conf** for a pseudo device driver that implements a RAM disk. This file creates two nodes called "ramdisk". The first entry creates ramdisk node instance 0, and the second creates ramdisk node, instance 1, with the additional disk-size property set to 512.

```
#
# Copyright (c) 1993, by Sun Microsystems, Inc.
#
#ident  "@(#)ramdisk.conf        1.3     93/06/04 SMI"

name="ramdisk" parent="pseudo" instance=0;
name="ramdisk" parent="pseudo" instance=1 disk-size=512;
```

SEE ALSO

driver.conf(4), **ddi_prop_op**(9F)

Writing Device Drivers

NAME

publickey – public key database

SYNOPSIS

/etc/publickey

DESCRIPTION

/etc/publickey is a local public key database that is used for secure RPC. The **/etc/publickey** file can be used in conjunction with or instead of other publickey databases, including the NIS publickey map and the NIS+ publickey map. Each entry in the database consists of a network user name (which may refer to either a user or a hostname), followed by the user's public key (in hex notation), a colon, and then the user's secret key encrypted with a password (also in hex notation).

The **/etc/publickey** file contains a default entry for **nobody**.

SEE ALSO

chkey(1), **newkey**(1M), **getpublickey**(3N), **nsswitch.conf**(4)

NAME

queuedefs – queue description file for at, batch, and cron

SYNOPSIS

/etc/cron.d/queuedefs

DESCRIPTION

The **queuedefs** file describes the characteristics of the queues managed by **cron**(1M). Each non-comment line in this file describes one queue. The format of the lines are as follows:

*q***.**[*njob***j**][*nice***n**][*nwait***w**]

The fields in this line are:

q The name of the queue. **a** is the default queue for jobs started by **at**(1); **b** is the default queue for jobs started by **batch** (see **at**(1)); **c** is the default queue for jobs run from a **crontab**(1) file.

njob The maximum number of jobs that can be run simultaneously in that queue; if more than *njob* jobs are ready to run, only the first *njob* jobs will be run, and the others will be run as jobs that are currently running terminate. The default value is **100**.

nice The **nice**(1) value to give to all jobs in that queue that are not run with a user ID of super-user. The default value is **2**.

nwait The number of seconds to wait before rescheduling a job that was deferred because more than *njob* jobs were running in that job's queue, or because the system-wide limit of jobs executing has been reached. The default value is **60**.

Lines beginning with **#** are comments, and are ignored.

EXAMPLES

```
#
# @(#)queuedefs.4 1.9 94/03/02 SMI; from S5R4
#
a.4j1n
b.2j2n90w
```

This file specifies that the **a** queue, for **at** jobs, can have up to 4 jobs running simultaneously; those jobs will be run with a **nice** value of 1. As no *nwait* value was given, if a job cannot be run because too many other jobs are running **cron** will wait 60 seconds before trying again to run it.

The **b** queue, for **batch**(1) jobs, can have up to 2 jobs running simultaneously; those jobs will be run with a **nice**(1) value of 2. If a job cannot be run because too many other jobs are running, **cron**(1M) will wait 90 seconds before trying again to run it. All other queues can have up to 100 jobs running simultaneously; they will be run with a **nice** value of 2, and if a job cannot be run because too many other jobs are running **cron** will wait 60 seconds before trying again to run it.

FILES

/etc/cron.d/queuedefs queue description file for **at**, **batch**, and **cron**.

SEE ALSO **at**(1), **nice**(1), **crontab**(1), **cron**(1M)

NAME

remote – remote host description file

SYNOPSIS

/etc/remote

DESCRIPTION

The systems known by **tip**(1) and their attributes are stored in an ASCII file which is structured somewhat like the **termcap** file. Each line in the file provides a description for a single *system*. Fields are separated by a colon '**:**'. Lines ending in a '\' character with an immediately following NEWLINE are continued on the next line.

The first entry is the name(s) of the host system. If there is more than one name for a system, the names are separated by vertical bars. After the name of the system comes the fields of the description. A field name followed by an '**=**' sign indicates a string value follows. A field name followed by a '**#**' sign indicates a following numeric value.

Entries named **tip***baudrate* are used as default entries by **tip**, as follows. When **tip** is invoked with only a phone number, it looks for an entry of the form **tip***baudrate***,** where *baudrate* is the baud rate with which the connection is to be made. For example, if the connection is to be made at **300** baud, **tip** looks for an entry of the form **tip300**.

CAPABILITIES

Capabilities are either strings **(str)**, numbers **(num)**, or boolean flags **(bool)**. A string capability is specified by *capability=value*; for example, '**dv=/dev/harris**'. A numeric capability is specified by *capability#value*; for example, '**xa#99**'. A boolean capability is specified by simply listing the capability.

at **(str)** Auto call unit type. The following lists valid '**at**' types and their corresponding hardware:

biz31f	Bizcomp 1031, tone dialing
biz31w	Bizcomp 1031, pulse dialing
biz22f	Bizcomp 1022, tone dialing
biz22w	Bizcomp 1022, pulse dialing
df02	DEC DF02
df03	DEC DF03
ventel	Ventel 212+
v3451	Vadic 3451 Modem
v831	Vadic 831
hayes	Any Hayes-compatible modem
at	Any Hayes-compatible modem

br **(num)** The baud rate used in establishing a connection to the remote host. This is a decimal number. The default baud rate is **300** baud.

cm **(str)** An initial connection message to be sent to the remote host. For example, if a host is reached through a port selector, this might be set to the appropriate sequence required to switch to the host.

cu **(str)** Call unit if making a phone call. Default is the same as the **dv** field.

db **(bool)** Cause **tip**(1) to ignore the first hangup it sees. **db** (dialback) allows the user to remain in **tip** while the remote machine disconnects and places a call back to the local machine. For more information about dialback configuration, see

TCP/IP Network Administration Guide.

di **(str)** Disconnect message sent to the host when a disconnect is requested by the user.

du **(bool)** This host is on a dial-up line.

dv **(str)** Device(s) to open to establish a connection. If this file refers to a terminal line, **tip** attempts to perform an exclusive open on the device to insure only one user at a time has access to the port.

ec **(bool)** Initialize the **tip** variable **echocheck** to *on*, so that **tip** will synchronize with the remote host during file transfer by waiting for the echo of the last character transmitted.

el **(str)** Characters marking an end-of-line. The default is no characters. **tip** only recognizes '**~**' escapes after one of the characters in **el**, or after a RETURN.

es **(str)** The command prefix (escape) character for **tip**.

et **(num)** Number of seconds to wait for an echo response when echo-check mode is on. This is a decimal number. The default value is **10** seconds.

ex **(str)** Set of non-printable characters not to be discarded when scripting with beautification turned on. The default value is **"\t\n\b\f"**.

fo **(str)** Character used to force literal data transmission. The default value is '**\377**'.

fs **(num)** Frame size for transfers. The default frame size is equal to **1024**.

hd **(bool)** Initialize the **tip** variable **halfduplex** to *on*, so local echo should be performed.

hf **(bool)** Initialize the **tip** variable **hardwareflow** to *on*, so hardware flow control is used.

ie **(str)** Input end-of-file marks. The default is a null string ("").

nb **(bool)** Initialize the **tip** variable **beautify** to *off*, so that unprintable characters will not be discarded when scripting.

nt **(bool)** Initialize the **tip** variable **tandem** to *off*, so that XON / XOFF flow control will not be used to throttle data from the remote host.

nv **(bool)** Initialize the **tip** variable **verbose** to *off*, so that verbose mode will be turned on.

oe **(str)** Output end-of-file string. The default is a null string (""). When **tip** is transferring a file, this string is sent at end-of-file.

pa **(str)** The type of parity to use when sending data to the host. This may be one of **even**, **odd**, **none**, **zero** (always set bit **8** to **0**), **one** (always set bit **8** to **1**). The default is **none**.

pn **(str)** Telephone number(s) for this host. If the telephone number field contains an '**@**' sign, **tip** searches the **/etc/phones** file for a list of telephone numbers — see **phones**(4). A '**%**' sign in the telephone number indicates a 5-second delay for the Ventel Modem.

For Hayes-compatible modems, if the telephone number starts with an 'S', the telephone number string will be sent to the modem without the "**DT**", which allows reconfiguration of the modem's S-registers and other parameters; for example, to disable auto-answer: "**pn=S0=0DT5551234**"; or to also restrict the modem to return only the basic result codes: "**pn=S0=0X0DT5551234**".

pr **(str)** Character that indicates end-of-line on the remote host. The default value is '\n'.

ra **(bool)** Initialize the **tip** variable **raise** to *on*, so that lower case letters are mapped to upper case before sending them to the remote host.

rc **(str)** Character that toggles case-mapping mode. The default value is '**\377**'.

re **(str)** The file in which to record session scripts. The default value is **tip.record**.

rw **(bool)** Initialize the **tip** variable **rawftp** to *on*, so that all characters will be sent as is during file transfers.

sc **(bool)** Initialize the **tip** variable **script** to *on*, so that everything transmitted by the remote host will be recorded.

tb **(bool)** Initialize the **tip** variable **tabexpand** to *on*, so that tabs will be expanded to spaces during file transfers.

tc **(str)** Indicates that the list of capabilities is continued in the named description. This is used primarily to share common capability information.

EXAMPLES

Here is a short example showing the use of the capability continuation feature:

```
UNIX-1200:\
        :dv=/dev/cua0:el=^D^U^C^S^Q^O@:du:at=ventel:ie=#$%:oe=^D:br#1200:
arpavax|ax:\
        :pn=7654321%:tc=UNIX-1200
```

FILES

/etc/remote	remote host description file.
/etc/phones	remote host phone number database.

SEE ALSO

tip(1), **phones**(4)

TCP/IP Network Administration Guide

NAME

resolv.conf – configuration file for name server routines

DESCRIPTION

The resolver configuration file contains information that is read by the resolver routines the first time they are invoked in a process. The file is designed to be human readable and contains a list of keyword-value pairs that provide various types of resolver information of the form:

keyword value

The different configuration options are:

nameserver *address* — The Internet address (in dot ('.') notation) of a name server that the resolver should query. At least one name server should be listed. Up to **MAXNS** (currently three) name servers may be listed, in that case the resolver library queries tries them in the order listed. The algorithm used is to try a name server, and if the query times out, try the next until out of name servers, then repeat trying all the name servers until a maximum number of retries are made.

domain *name* — The default domain to append to names that do not have a dot ('.') in them.

The keyword-value pair must appear on a single line, and the keyword (for instance, **nameserver**) must start the line. The value follows the keyword, separated by white space.

FILES

/etc/resolv.conf

SEE ALSO

in.named(1M), **gethostbyname**(3N), **resolver**(3N)

NAME

rmmount.conf – removable media mounter configuration file

SYNOPSIS

/etc/rmmount.conf

DESCRIPTION

The **rmmount.conf** file contains the **rmmount**(1M) configuration information. This file describes where to find shared objects that perform actions on file systems after identifying and mounting them. The **rmmount.conf** file is also used to share CD-ROM and floppy file systems.

Actions are executed in the order in which they appear in the configuration file. The action function can return either 1 or 0. If it returns 0, no further actions will be executed. This allows the function to control which applications are executed. For example, **action_filemgr** always returns 0 if the File Manager is running, thereby preventing subsequent actions from being executed.

To execute an action after media has been inserted and while the File Manager is not running, list the action after **action_filemgr** in the **rmmount.conf** file. To execute an action before the File Manager becomes aware of the media, list the action before **action_filemgr** in the **rmmount.conf file.**

The syntax for the **rmmount.conf** file is as follows.

 # File system identification
ident *filesystem_type shared_object media_type* [*media_type* . . .]

 # Actions
action *media_type shared_object args_to_so*

 # File system sharing
share *media_or_file_system share_command_options*

Explanations of the syntax for the **File system identification** fields are as follows.

filesystem_type An ASCII string used as the file system type flag of the **mount** command (see the **–F** option of **mount**(1M)). It is also used to match names passed to **rmmount**(1M) from Volume Management.

shared_object Programs that identify file systems and perform actions. This *shared_object* is found at **/usr/lib/fs/***filesystem_type* / *shared_object*.

media_type The type of media where this file system resides. Legal values are **cdrom** and **floppy**.

Explanations of the syntax for the **Actions** fields are as follows.

media_type Type of media. This argument is passed in from Volume Management as **VOLUME_TYPE.**

shared_object Programs that identify file systems and perform actions. If *shared_object* starts with ‘/’ (slash), the full path name is used; otherwise, **/usr/lib/rmmount** is prepended to the name.

args_to_so Arguments passed to the *shared_object*. These arguments are passed in as an *argc* and *argv*[].

The definition of the interface to **Actions** is located in **/usr/include/rmmount.h**.

Explanations of the syntax for the **File system sharing** fields are as follows.

media_or_file_system
: Either the type of media (CD-ROM or floppy) or the specific file system to share.

share_command_options
: Options of the **share** command. See **share**(1M) for more information about these options.

Default Values

The following is an example of an **rmmount.conf** file.

```
#
# Removable Media Mounter configuration file.
#
# File system identification
ident hsfs ident_hsfs.so cdrom
ident ufs ident_ufs.so cdrom floppy
ident pcfs ident_pcfs.so floppy
# Actions
action cdrom action_filemgr.so
action floppy action_filemgr.so
```

EXAMPLES

The following examples show how various file systems are shared using the share syntax for the **rmmount.conf** file. These lines are added after the Actions entries.

share cdrom*
: Shares all CD-ROMs via NFS and applies no access restrictions.

share solaris_2.x*
: Shares CD-ROMs named solaris_2.x* with no access restrictions.

share cdrom* -o ro=engineering
: Shares all CD-ROMs via NFS but exports only to the "engineering" net-group.

share solaris_2.x* -d distribution CD
: Shares CD-ROMs named solaris_2.x* with no access restrictions and with the description that it is a distribution CD-ROM.

share floppy0
: Shares any floppy inserted into floppy drive 0.

SEE ALSO

volcancel(1), **volcheck**(1), **volmissing**(1) **rmmount**(1M), **share**(1M), **vold**(1M), **vold.conf**(4), **volfs**(7),

NAME

rmtab – remote mounted file system table

SYNOPSIS

/etc/rmtab

DESCRIPTION

rmtab contains a table of filesystems that are remotely mounted by NFS clients. This file is maintained by **mountd**(1M), the mount daemon. The data in this file should be obtained only from **mountd**(1M) using the **MOUNTPROC_DUMP** remote procedure call.

The file contains a line of information for each remotely mounted filesystem. There are a number of lines of the form:

hostname:*fsname*

The mount daemon adds an entry for any client that successfully executes a mount request and deletes the appropriate entries for an unmount request.

Lines beginning with a hash (' **#**') are commented out. These lines are removed from the file by **mountd**(1M) when it first starts up. Stale entries may accumulate for clients that crash without sending an unmount request.

FILES

/etc/rmtab

SEE ALSO

mountd(1M), **showmount**(1M)

NAME

routing – system support for packet network routing

DESCRIPTION

The network facilities provide general packet routing. Routing table maintenance may be implemented in applications processes.

A simple set of data structures compose a "routing table" used in selecting the appropriate network interface when transmitting packets. This table contains a single entry for each route to a specific network or host. The routing table was designed to support routing for the Internet Protocol (IP), but its implementation is protocol independent and thus it may serve other protocols as well. User programs may manipulate this data base with the aid of two **ioctl**(2) commands, **SIOCADDRT** and **SIOCDELRT**. These commands allow the addition and deletion of a single routing table entry, respectively. Routing table manipulations may only be carried out by privileged user.

A routing table entry has the following form, as defined in **/usr/include/net/route.h**:

```
struct rtentry {
        u_long              rt_hash;     /* to speed lookups */
        struct sockaddr     rt_dst;      /* key */
        struct sockaddr     rt_gateway;  /* value */
        short               rt_flags;    /* up/down?, host/net */
        short               rt_refcnt;   /* # held references */
        u_long              rt_use;      /* raw # packets forwarded */
#ifdef STRNET
        struct ip_provider  *rt_prov;    /* the answer: provider to use */
#else
   struct  ifnet            *rt_ifp;     /* the answer: interface to use */
#endif /* STRNET*/
};
```

with *rt_flags* defined from:

```
#define  RTF_UP          0x1     /* route usable */
#define  RTF_GATEWAY     0x2     /* destination is a gateway */
#define  RTF_HOST        0x4     /* host entry (net otherwise) */
```

Routing table entries come in three flavors: for a specific host, for all hosts on a specific network, for any destination not matched by entries of the first two types (a wildcard route). Each network interface installs a routing table entry when it it is initialized. Normally the interface specifies the route through it is a "direct" connection to the destination host or network. If the route is direct, the transport layer of a protocol family usually requests the packet be sent to the same host specified in the packet. Otherwise, the interface may be requested to address the packet to an entity different from the eventual recipient (that is, the packet is forwarded).

Routing table entries installed by a user process may not specify the hash, reference count, use, or interface fields; these are filled in by the routing routines. If a route is in use when it is deleted (**rt_refcnt** is non-zero), the resources associated with it will not be reclaimed until all references to it are removed.

User processes read the routing tables through the **/dev/ip** device.

The *rt_use* field contains the number of packets sent along the route. This value is used to select among multiple routes to the same destination. When multiple routes to the same destination exist, the least used route is selected.

A wildcard routing entry is specified with a zero destination address value. Wildcard routes are used only when the system fails to find a route to the destination host and network. The combination of wildcard routes and routing redirects can provide an economical mechanism for routing traffic.

FILES

/dev/ip	IP device driver

SEE ALSO

route(1M), **ioctl**(2)

DIAGNOSTICS

EEXIST	A request was made to duplicate an existing entry.
ESRCH	A request was made to delete a non-existent entry.
ENOBUFS	Insufficient resources were available to install a new route.

NAME

rpc – rpc program number data base

SYNOPSIS

/etc/rpc

DESCRIPTION

The **rpc** file is a local source containing user readable names that can be used in place of RPC program numbers. The **rpc** file can be used in conjunction with or instead of other rpc sources, including the NIS maps ''rpc.byname'' and ''rpc.bynumber'' and the NIS+ table ''rpc''.

The rpc file has one line for each RPC program name. The line has the following format:

name-of-the-RPC-program RPC-program-number aliases

Items are separated by any number of blanks and/or tab characters. A ''**#**'' indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

EXAMPLES

Below is an example of an RPC database:

```
#
#              rpc
#
rpcbind        100000          portmap     sunrpc      portmapper
rusersd        100002          rusers
nfs            100003          nfsprog
mountd         100005          mount       showmount
walld          100008          rwall       shutdown
sprayd         100012          spray
llockmgr       100020
nlockmgr       100021
status         100024
bootparam      100026
keyserv        100029          keyserver
```

FILES

/etc/nsswitch.conf

SEE ALSO

nsswitch.conf(4)

NAME

rpld.conf – Remote Program Load (RPL) server configuration file

SYNOPSIS

/etc/rpld.conf

AVAILABILITY

x86

DESCRIPTION

The **/etc/rpld.conf** file contains the configuration information for operation of **rpld**, the RPL-based network boot server. It is a text file containing keyword-value pairs and comments. The keyword-value pairs specify the value to use for parameters used by the RPL server. Comments can be entered by starting the line using the # character. The user can add comments to the file for customized configurations. Alternate RPL server configuration files can be specified when running the RPL server by supplying a configuration file similar to the default configuration file.

Keywords

All keywords are case-sensitive. Not all keywords must be present. (However, note that the **end** keyword at the end of the file must be present.) If a keyword is not present, internal defaults, which are the default values described here, will be used. Keyword-value pairs are specified by:

keyword = value

DebugLevel

Specify the number of error, warning, and information messages to be generated while the RPL server is running. The valid range is 0-9. A value of 0 means no message at all, while a value of 9 will generate the most messages. The default is 0. Note that it is best to limit the value to 8 or below; use of level 9 may generate so many debug messages that the performance of the RPL server may be impacted.

DebugDest

A numeric value specifying where to send the messages to:

0 = standard output
1 = syslogd
2 = log file

The default is 2.

MaxClients

A numeric value specifying the maximum number of simultaneous network boot clients to be in service. A value of –1 means unlimited except where system resources is the limiting factor. Any positive value will set a limit on the number of clients to be in service at the same time unless system resource constraints come in before the limit. The default is –1.

BackGround

A numeric value indicating whether the RPL server should run in the background or not. A 0 means run in the background and a 1 means do not run in the background. The difference is whether the server will relinquish the controlling terminal or not. The default is 1.

FrameSize
The default size of data frames to be used to send bootfile data to the network boot clients. This size should not exceed the limits imposed by the underlying physical media. For **ethernet/802.3**, the maximum physical frame size is 1500 octets. The default is 1500. Note that the protocol overhead of LLC1 and RPL is 32 octets, resulting in a maximum data length of 1468 octets.

LogFile The log file to which messages will be sent if **DebugDest** is set to 2 (the default). The default file is **var/spool/rpld.log**.

StartDelay
The initial delay factor to use to control the speed of downloading. In the default mode of operation, the downloading process does not wait for a positive acknowledgment from the client before the next data frame is sent. In the case of a fast server and slow client, data overrun can result and requests for retransmission will be frequent. By using a delay factor, the speed of data transfer is controlled to avoid retransmission requests. Note that the unit of delay is machine dependent and bears no correlation with the actual time delayed.

DelayGran
Delay granularity. If the initial delay factor is not suitable and the rate of downloading is either too fast or too slow, retransmission requests from the clients will be used to adjust the delay factor either upward (to slow down the data rate) or downward (to speed up the data rate). The delay granularity is used as the delay delta for adjustment.

end Keyword at the end of the file. It must be present.

FILES

/etc/rpld.conf
/usr/sbin/rpld

SEE ALSO

rpld(1M)

NAME

rt_dptbl – real-time dispatcher parameter table

DESCRIPTION

The process scheduler (or dispatcher) is the portion of the kernel that controls allocation of the CPU to processes. The scheduler supports the notion of scheduling classes where each class defines a scheduling policy, used to schedule processes within that class. Associated with each scheduling class is a set of priority queues on which ready to run processes are linked. These priority queues are mapped by the system configuration into a set of global scheduling priorities which are available to processes within the class. (The dispatcher always selects for execution the process with the highest global scheduling priority in the system.) The priority queues associated with a given class are viewed by that class as a contiguous set of priority levels numbered from 0 (lowest priority) to *n* (highest priority—a configuration dependent value). The set of global scheduling priorities that the queues for a given class are mapped into might not start at zero and might not be contiguous (depending on the configuration).

The real-time class maintains an in-core table, with an entry for each priority level, giving the properties of that level. This table is called the real-time dispatcher parameter table (**rt_dptbl**). The **rt_dptbl** consists of an array (**config_rt_dptbl[]**) of parameter structures (**struct rtdpent_t**), one for each of the *n* priority levels. The structure are accessed via a pointer, (**rt_dptbl**), to the array. The properties of a given priority level *i* are specified by the *i*th parameter structure in this array (**rt_dptbl[***i***]**).

A parameter structure consists of the following members. These are also described in the **/usr/include/sys/rt.h** header file.

rt_globpri The global scheduling priority associated with this priority level. The **rt_globpri** values cannot be changed with **dispadmin**(1M).

rt_quantum The length of the time quantum allocated to processes at this level in ticks (Hz). The time quantum value is only a default or starting value for processes at a particular level as the time quantum of a real-time process can be changed by the user with the **priocntl** command or the **priocntl** system call.

An administrator can affect the behavior of the real-time portion of the scheduler by reconfiguring the **rt_dptbl**. There are two methods available for doing this: reconfigure with a loadable module at boot-time or by using **dispadmin**(1M) at run-time.

RT_DPTBL LOADABLE MODULE

The **rt_dptbl** can be reconfigured with a loadable module which contains a new real time dispatch table. The module containing the dispatch table is separate from the RT loadable module which contains the rest of the real time software. This is the only method that can be used to change the number of real time priority levels or the set of global scheduling priorities used by the real time class. The relevant procedure and source code is described in the **REPLACING THE RT_DPTBL LOADABLE MODULE** section.

DISPADMIN CONFIGURATION FILE

The **rt_quantum** values in the **rt_dptbl** can be examined and modified on a running system using the **dispadmin**(1M) command. Invoking **dispadmin** for the real-time class allows the administrator to retrieve the current **rt_dptbl** configuration from the kernel's in-core table, or overwrite the in-core table with values from a configuration file. The configuration file used for input to **dispadmin** must conform to the specific format described below.

Blank lines are ignored and any part of a line to the right of a # symbol is treated as a comment. The first non-blank, non-comment line must indicate the resolution to be used for interpreting the time quantum values. The resolution is specified as

RES=*res*

where *res* is a positive integer between 1 and 1,000,000,000 inclusive and the resolution used is the reciprocal of *res* in seconds. (For example, **RES=1000** specifies millisecond resolution.) Although very fine (nanosecond) resolution may be specified, the time quantum lengths are rounded up to the next integral multiple of the system clock's resolution.

The remaining lines in the file are used to specify the **rt_quantum** values for each of the real-time priority levels. The first line specifies the quantum for real-time level 0, the second line specifies the quantum for real-time level 1, etc. There must be exactly one line for each configured real-time priority level. Each **rt_quantum** entry must be either a positive integer specifying the desired time quantum (in the resolution given by *res*), or the value -2 indicating an infinite time quantum for that level.

EXAMPLES

The following excerpt from a **dispadmin** configuration file illustrates the format. Note that for each line specifying a time quantum there is a comment indicating the corresponding priority level. These level numbers indicate priority within the real-time class, and the mapping between these real-time priorities and the corresponding global scheduling priorities is determined by the configuration specified in the **RT_DPTBL** loadable module. The level numbers are strictly for the convenience of the administrator reading the file and, as with any comment, they are ignored by **dispadmin** on input. **dispadmin** assumes that the lines in the file are ordered by consecutive, increasing priority level (from 0 to the maximum configured real-time priority). The level numbers in the comments should normally agree with this ordering; if for some reason they don't, however, **dispadmin** is unaffected.

```
# Real-Time Dispatcher Configuration File
RES=1000
#  TIME QUANTUM     PRIORITY
   # (rt_quantum)      LEVEL
         100            #  0
         100            #  1
         100            #  2
         100            #  3
         100            #  4
         100            #  5
          90            #  6
          90            #  7
           .            .  .
           .            .  .
           .            .  .
          10            #  58
          10            #  59
```

REPLACING THE RT_DPTBL LOADABLE MODULE

In order to change the size of the real time dispatch table, the loadable module which contains the dispatch table information will have to be built. It is recommended that you save the existing module before using the following procedure.

1. Place the dispatch table code shown below in a file called **rt_dptbl.c** An example of an **rt_dptbl.c** file follows.
2. Compile the code using the given compilation and link lines supplied.

   cc –c –0 –D_KERNEL rt_dptbl.c
   ld –r –o RT_DPTBL rt_dptbl.o

3. Copy the current dispatch table in **/usr/kernel/sched** to **RT_DPTBL.bak**.
4. Replace the current **RT_DPTBL** in **/usr/kernel/sched**.
5. You will have to make changes in the **/etc/system** file to reflect the changes to the sizes of the tables. See **system**(4). The **rt_maxpri** variable may need changing. The syntax for setting this is:

   set RT:rt_maxpri=(class-specific value for maximum real-time priority)

6. Reboot the system to use the new dispatch table.

NOTE: Great care should be used in replacing the dispatch table using this method. If you don't get it right, the system may not behave properly. The following is an example of a **rt_dptbl.c** file used for building the new **rt_dptbl**.

```
                /* BEGIN rt_dptbl.c */

#include <sys/proc.h>
#include <sys/priocntl.h>
#include <sys/class.h>
#include <sys/disp.h>
#include <sys/rt.h>
#include <sys/rtpriocntl.h>

/*
 * This is the loadable module wrapper.
 */
#include <sys/modctl.h>

extern struct mod_ops mod_miscops;

/*
 * Module linkage information for the kernel.
 */

static struct modlmisc modlmisc = {
                &mod_miscops, "realtime dispatch table"
};

static struct modlinkage modlinkage = {
                MODREV_1, &modlmisc, 0
};

_init()
{
                return (mod_install(&modlinkage));
}

_info (struct modinfo *modinfop)
{
                return (mod_info(&modlinkage, modinfop));
}


rtdpent_t     config_rt_dptbl[] = {
/*   prilevel     Time quantum */
     100,                100,
     101,                100,
     102,                100,
     103,                100,
```

```
104,    100,
105,    100,
106,    100,
107,    100,
108,    100,
109,    100,
110,     80,
111,     80,
112,     80,
113,     80,
114,     80,
115,     80,
116,     80,
117,     80,
118,     80,
119,     80,
120,     60,
121,     60,
122,     60,
123,     60,
124,     60,
125,     60,
126,     60,
127,     60,
128,     60,
129,     60,
130,     40,
131,     40,
132,     40,
133,     40,
134,     40,
135,     40,
136,     40,
137,     40,
138,     40,
139,     40,
140,     20,
141,     20,
142,     20,
143,     20,
144,     20,
145,     20,
146,     20,
147,     20,
148,     20,
```

```
    149,            20,
    150,            10,
    151,            10,
    152,            10,
    153,            10,
    154,            10,
    155,            10,
    156,            10,
    157,            10,
    158,            10,
    159,            10,
};

/*
 * Return the address of config_rt_dptbl
 */
rtdpent_t *
rt_getdptbl()
{
        return (config_rt_dptbl);
}
```

FILES

<sys/rt.h>

SEE ALSO

priocntl(1), **dispadmin**(1M), **priocntl**(2), **system**(4)

File System Administration
System Services Guide

NAME

sbus – configuration files for SBus device drivers

AVAILABILITY

SPARC

DESCRIPTION

The SBus is a geographically addressed peripheral bus present on many SPARC hardware platforms. SBus devices are *self-identifying* — that is to say the SBus card itself provides information to the system so that it can identify the device driver that needs to be used. The device usually provides additional information to the system in the form of name-value pairs that can be retrieved using the DDI property interfaces. See **ddi_prop_op**(9F) for details.

The information is usually derived from a small Forth program stored in the FCode PROM on the card, so driver configuration files should be completely unnecessary for these devices. However, on some occasions, drivers for SBus devices may need to use driver configuration files to augment the information provided by the SBus card. See **driver.conf**(4) for further details.

When they are needed, configuration files for SBus device drivers should identify the parent bus driver implicitly using the *class* keyword. This removes the dependency on the particular bus driver involved since this may be named differently on different platforms.

All bus drivers of class **sbus** recognise the following properties:

reg An arbitrary length array where each element of the array consists of a 3-tuple of integers. Each array element describes a logically contiguous mappable resource on the SBus.

The first integer of each tuple specifies the slot number the card is plugged into. The second integer of each 3-tuple specifies the offset in the slot address space identified by the first element. The third integer of each 3-tuple specifies the size in bytes of the mappable resource.

The driver can refer to the elements of this array by index, and construct kernel mappings to these addresses using **ddi_map_regs**(9F). The index into the array is passed as the *rnumber* argument of **ddi_map_regs( )**.

interrupts An arbitrary length array where each element of the array consists of a single integer. Each array element describes a possible SBus interrupt level that the device might generate.

The driver can refer to the elements of this array by index, and register interrupt handlers with the system using **ddi_add_intr**(9F). The index into the array is passed as the *inumber* argument of **ddi_add_intr( )**.

intr An arbitrary length array where each element of the array consists of a pair of integers. The first element of each pair specifies the SPARC interrupt priority level, the second element is a vector number which should be specified as zero.

This property is recognised for compatibility with older SBus cards. The **interrupts** property overrides any interrupt specification in the **intr**

property.

registers An arbitrary length array where each element of the array consists of a 3-tuple of integers. Each array element describes a logically contiguous mappable resource on the SBus.

The first integer of each tuple should be set to **–1**, specifying that any SBus slot may be matched. The second integer of each 3-tuple specifies the offset in the slot address space identified by the first element. The third integer of each 3-tuple specifies the size in bytes of the mappable resoure.

The **registers** property can only be used to augment an incompletely specified **reg** property with information from a driver configuration file. It may only be specified in a driver configuration file.

All SBus devices must provide **reg** properties to the system. The first two integer elements of the **reg** property are used to construct the address part of the device name under **/devices**.

Only devices that generate interrupts need to provide **interrupts** (or **intr**) properties.

Occasionally, it may be necessary to override or augment the configuration information supplied by the SBus device. This can be achieved by writing a driver configuration file that describes a prototype device information (devinfo) node specification, containing the additional properties required.

For the system to merge the information, certain conditions must be met. First, the **name** property must be the same. Second, either the first two integers (slot number and offset) of the two **reg** properties must be the same, or the second integer (offset) of the **reg** and **registers** properties must be the same.

In the event that the SBus card has no **reg** property at all, the self-identifying information cannot be used, so all the details of the card must be specified in a driver configuration file.

EXAMPLES

Here is a configuration file for an SBus card called **SUNW,netboard**. The card already has a simple FCode PROM that creates **name** and **reg** properties, and will have a complete set of properties for normal use once the driver and firmware is complete.

In this example, we want to augment the properties given to us by the firmware. We use the same **name** property, and use the **registers** property to match the firmware **reg** property. That way we don't have to worry about which slot the card is really plugged into.

We want to add an **interrupts** property while we are developing the firmware and driver so that we can start to experiment with interrupts. The device can generate interrupts at SBus level 3. Additionally, we want to set a **debug-level** property to 4.

```
#
# Copyright (c) 1992, by Sun Microsystems, Inc.
#
#ident  "@(#)SUNW,netboard.conf       1.4     92/03/10 SMI"
```

```
name="SUNW,netboard" class="sbus"
        registers=-1,0x40000,64,-1,0x80000,1024
        interrupts=3 debug-level=4;
```

SEE ALSO **driver.conf**(4), **ddi_add_intr**(9F), **ddi_map_regs**(9F), **ddi_prop_op**(9F)

Writing Device Drivers

WARNINGS The wildcarding mechanism of the **registers** property matches every instance of the particular device attached to the system. This may not always be what is wanted.

NAME

sccsfile – format of an SCCS history file

DESCRIPTION

An SCCS file is an ASCII file consisting of six logical parts:

checksum	character count used for error detection
delta table	log containing version info and statistics about each delta
usernames	login names and / or group IDs of users who may add deltas
flags	definitions of internal keywords
comments	arbitrary descriptive information about the file
body	the actual text lines intermixed with control lines

Each section is described in detail below.

Conventions

Throughout an SCCS file there are lines which begin with the ASCII SOH (start of heading) character (octal 001). This character is hereafter referred to as the *control character*, and will be represented as '^A'. If a line described below is not depicted as beginning with the control character, it cannot do so and still be within SCCS file format.

Entries of the form *ddddd* represent a five digit string (a number between 00000 and 99999).

Checksum

The checksum is the first line of an SCCS file. The form of the line is:

^A **h***ddddd*

The value of the checksum is the sum of all characters, except those contained in the first line. The ^A**h** provides a *magic number* of (octal) 064001.

Delta Table

The delta table consists of a variable number of entries of the form:

^A**s** *inserted/deleted/unchanged*
^A**d** *type sid yr/mo/da hr:mi:se username serial-number predecessor-sn*
^*A***i** *include-list*
^A**x** *exclude-list*
^A**g** *ignored-list*
^A**m** *mr-number*
...
^A**c** *comments ...*
...
^A**e**

The first line (^A**s**) contains the number of lines inserted / deleted / unchanged respectively. The second line (^A**d**) contains the type of the delta (normal: **D**, and removed: **R**), the SCCS ID of the delta, the date and time of creation of the delta, the user-name corresponding to the real user ID at the time the delta was created, and the serial numbers of the delta and its predecessor, respectively.

The ^A**i**, ^A**x**, and ^A**g** lines contain the serial numbers of deltas included, excluded, and ignored, respectively. These lines do not always appear.

The ^A**m** lines (optional) each contain one MR number associated with the delta; the ^A**c** lines contain comments associated with the delta.

The ^A**e** line ends the delta table entry.

User Names

The list of user-names and / or numerical group IDs of users who may add deltas to the file, separated by NEWLINE characters. The lines containing these login names and / or numerical group IDs are surrounded by the bracketing lines ^A**u** and ^A**U**. An empty list allows anyone to make a delta.

Flags

Flags are keywords that are used internally (see **sccs-admin**(1) for more information on their use). Each flag line takes the form:

^A**f** *flag optional text*

The following flags are defined in order of appearance:

^A**f** t *type-of-program*
Defines the replacement for the **%T%** ID keyword.

^A**f** v program-name
Controls prompting for MR numbers in addition to comments; if the optional text is present it defines an MR number validity checking program.

^A**f** i Indicates that the '**No id keywords**' message is to generate an error that terminates the SCCS command. Otherwise, the message is treated as a warning only.

^A**f** b Indicates that the –**b** option may be used with the SCCS **get** command to create a branch in the delta tree.

^A**f** m module name
Defines the first choice for the replacement text of the **%M%** ID keyword.

^A**f** f *floor*
Defines the "floor" release; the release below which no deltas may be added.

^A**f** c *ceiling*
Defines the "ceiling" release; the release above which no deltas may be added.

^A**f** d *default-sid*
The **d** flag defines the default SID to be used when none is specified on an SCCS **get** command.

^A**f** n The **n** flag enables the SCCS **delta** command to insert a "null" delta (a delta that applies *no* changes) in those releases that are skipped when a delta is made in a *new* release (for example, when delta 5.1 is made after delta 2.7, releases 3 and 4 are skipped).

^A**f** j Enables the SCCS **get** command to allow concurrent edits of the same base SID.

^A**f** l *lock-releases*
Defines a *list* of releases that are locked against editing.

^A**f** q user defined
Defines the replacement for the **%Q%** ID keyword.

^A**f** e **0** | **1**
The **e** flag indicates whether a source file is encoded or not. A **1** indicates that the file is encoded. Source files need to be encoded when they contain control characters, or when they do not end with a NEWLINE. The **e** flag allows files that contain binary data to be checked in.

Comments

Arbitrary text surrounded by the bracketing lines ^A**t** and ^A**T**. The comments section typically will contain a description of the file's purpose.

Body

The body consists of text lines and control lines. Text lines do not begin with the control character, control lines do. There are three kinds of control lines: *insert*, *delete*, and *end*, represented by:

^A**I** *ddddd*
^A**D** *ddddd*
^A**E** *ddddd*

respectively. The digit string is the serial number corresponding to the delta for the control line.

SEE ALSO

sccs(1), **sccs-admin**(1), **sccs-cdc**(1), **sccs-comb**(1), **sccs-delta**(1), **sccs-get**(1), **sccs-help**(1), **sccs-prs**(1), **sccs-prt**(1), **sccs-rmdel**(1), **sccs-sact**(1), **sccs-sccsdiff**(1), **sccs-unget**(1), **sccs-val**(1), **what**(1)

NAME

scsi – configuration files for SCSI target drivers

DESCRIPTION

The architecture of the Solaris SCSI subsystem distinguishes two types of device drivers: SCSI target drivers, and SCSI host adapter drivers. Target drivers like **sd**(7) and **st**(7) on SPARC and **cmdk**(7) and **cmtp**(7) on x86 manage the device on the other end of the SCSI bus. Host adapter drivers manage the SCSI bus on behalf of all the devices that share it.

Drivers for host adapters provide a common set of interfaces for target drivers. These interfaces comprise the Sun Common SCSI Architecture (SCSA) which are documented as part of the Solaris DDI / DKI. See **scsi_ifgetcap**(9F), **scsi_pktalloc**(9F), and **scsi_transport**(9F) for further details of these, and associated routines.

Target drivers for SCSI devices should use a driver configuration file to enable them to be recognized by the system.

Configuration files for SCSI target drivers should identify the host adapter driver implicitly using the *class* keyword to remove any dependency on the particular host adapter involved.

All host adapter drivers of class **scsi** recognise the following properties:

target Integer-valued SCSI target identifier that this driver will claim.

lun Integer-valued SCSI logical unit number (LUN) that this driver will claim.

All SCSI target drivers must provide **target** and **lun** properties. These properties are used to construct the address part of the device name under **/devices**.

EXAMPLES

Here is a configuration file for a SCSI target driver called **toaster.conf**.

```
#
# Copyright (c) 1992, by Sun Microsystems, Inc.
#
#ident  "@(#)toaster.conf          1.2     92/05/12 SMI"

name="toaster" class="scsi" target=4 lun=0;
```

SEE ALSO

driver.conf(4), **scsi_ifgetcap**(9F), **scsi_pktalloc**(9F), **scsi_transport**(9F)

Writing Device Drivers

ANSI Small Computer System Interface-2 (SCSI-2)

SPARC only **sd**(7), **st**(7)

x86 only **cmdk**(7), **cmtp**(7)

NOTES

You need to ensure that the **target** and **lun** values claimed by your target driver do not conflict with existing target drivers on the system. For example, on SPARC, if the target is a direct access device, the standard **sd.conf** file will usually make **sd** claim it before any other driver has a chance to probe it. This is also true for x86; if the target is a direct access device, the standard **cmdk.conf** file will usually make **cmdk** claim it before any other driver has a chance to probe it.

NAME

services – Internet services and aliases

SYNOPSIS

/etc/inet/services
/etc/services

DESCRIPTION

The **services** file is a local source of information regarding each service available through the Internet. The services file can be used in conjunction with or instead of other services sources, including the NIS maps "services.byname" and the NIS+ table "services." Programs use the **getservbyname**(3N) routines to access this information.

The services file contains an entry for each service. Each entry has the form:

service-name port/protocol aliases

service-name — This is the official Internet service name.

port / protocol — This field is composed of the port number and protocol through which the service is provided (for instance, **512/tcp**).

aliases — This is a list of alternate names by which the service might be requested.

Fields can be separated by any number of SPACE and / or TAB characters. A '**#**' (number sign) indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Service names may contain any printable character other than a field delimiter, NEWLINE, or comment character.

FILES

/etc/nsswitch.conf configuration file for name-service switch

SEE ALSO

getservbyname(3N), **inetd.conf**(4), **nsswitch.conf**(4)

NOTES

/etc/inet/services is the official SVR4 name of the **services** file. The symbolic link **/etc/services** exists for BSD compatibility.

NAME

shadow – shadow password file

DESCRIPTION

/etc/shadow is an access-restricted ASCII system file that stores users' encrypted passwords and related information. The shadow file can be used in conjunction with other shadow sources, including the NIS maps **passwd.byname** and **passwd.byuid** and the NIS+ table **passwd**. Programs use the **getspnam**(3C) routines to access this information.

The fields for each user entry are separated by colons. Each user is separated from the next by a newline. Unlike the **/etc/passwd** file, **/etc/shadow** does not have general read permission.

Each entry in the shadow file has the form:

username:*password*:*lastchg*:*min*:*max*:*warn*:*inactive*:*expire*:*flag*

The fields are defined as follows:

username	The user's login name (UID).
password	A 13-character encrypted password for the user, a *lock* string to indicate that the login is not accessible, or no string, which shows that there is no password for the login.
lastchg	The number of days between January 1, 1970, and the date that the password was last modified.
min	The minimum number of days required between password changes.
max	The maximum number of days the password is valid.
warn	The number of days before password expires that the user is warned.
inactive	The number of days of inactivity allowed for that user.
expire	An absolute date specifying when the login may no longer be used.
flag	Reserved for future use, set to zero. Currently not used.

The encrypted password consists of 13 characters chosen from a 64-character alphabet (**.**, **/**, **0–9**, **A–Z**, **a–z**). To update this file, use the **passwd**(1), **useradd**(1M), **usermod**(1M), or **userdel**(1M) commands.

In order to make system administration manageable, **/etc/shadow** entries should appear in exactly the same order as **/etc/passwd** entries; this includes ''+'' and ''-'' entries if the **compat** source is being used (see **nsswitch.conf**(4)).

FILES

/etc/shadow
/etc/passwd
/etc/nsswitch.conf

SEE ALSO

login(1), **passwd**(1), **useradd**(1M), **usermod**(1M), **userdel**(1M), **putspent**(3C), **getspnam**(3C), **nsswitch.conf**(4), **passwd**(4)

NAME

sharetab – shared file system table

DESCRIPTION

sharetab resides in directory **/etc/dfs** and contains a table of local resources shared by the **share** command.

Each line of the file consists of the following fields:

pathname resource fstype specific_options description

where

pathname	Indicate the path name of the shared resource.
resource	Indicate the symbolic name by which remote systems can access the resource.
fstype	Indicate the file system type of the shared resource.
specific_options	Indicate file-system-type-specific options that were given to the **share** command when the resource was shared.
description	Describe the shared resource provided by the system administrator when the resource was shared.

SEE ALSO

share(1M)

NAME

space – disk space requirement file

DESCRIPTION

space is an ASCII file that gives information about disk space requirements for the target environment. **space** defines space needed beyond what is used by objects defined in the **prototype** file; for example, files which will be installed with the **installf** command. **space** should define the maximum amount of additional space that a package will require.

The generic format of a line in this file is:

pathname blocks inodes

Definitions for the fields are as follows:

pathname Specify a directory name which may or may not be the mount point for a filesystem. Names that do not begin with a slash ('/') indicate relocatable directories.

blocks Define the number of disk blocks required for installation of the files and directory entries contained in the pathname (using a 512-byte block size).

inodes Define the number of inodes required for installation of the files and directory entries contained in the pathname.

EXAMPLES

```
# extra space required by config data which is
# dynamically loaded onto the system
data    500     1
```

SEE ALSO

installf(1M), **prototype**(4)

NAME

strftime – language specific strings

DESCRIPTION

Each locale has a printable file specifying date and time formatting information, **/usr/lib/locale/***locale***/LC_TIME** where *locale* is the locale name. These files specify:

1. abbreviated month names (in order)
2. full month names (in order)
3. abbreviated weekday names (in order)
4. full weekday names (in order)
5. string to specify local time representation (**%X**)
6. string to specify local date representation (**%x**)
7. string to specify local date and time (**%c**) for **strftime()** default
8. AM (ante meridian) string
9. PM (post meridian) string
10. string to specify local date and time (**%C**) for **cftime()** default

Each string is on a line by itself. All white space is significant. The order of the strings in the above list is the same order in which they must appear in the file.

EXAMPLES

/usr/lib/locale/C/LC_TIME

```
Jan
Feb
 . . .
January
February
 . . .
Sun
Mon
 . . .
Sunday
Monday
 . . .
%H:%M:%S
%m/%d/%y
%a %b %d %H:%M:%S %Y
AM
PM
%a %b %e %T %Z %Y
```

FILES

/usr/lib/locale/locale/LC_TIME

SEE ALSO **ctime**(3C), **setlocale**(3C), **strftime**(3C)

NOTES Do not change files under the C locale, as this could cause undefined or nonstandard behavior.

NAME

sysbus, isa, eisa, mca – configuration files for ISA, EISA, and MCA bus device drivers

AVAILABILITY

x86

DESCRIPTION

Solaris for x86 platforms support the ISA, EISA, and MCA buses as the system bus. Drivers for devices on these buses use driver configuration files to inform the system that the device hardware may be present (see **driver.conf**(4) for more information). The configuration file must specify the device I/O port addresses, any interrupt capabilities that the device may have, any DMA channels it may require, and any memory-mapped addresses it may occupy.

Configuration files for ISA, EISA, and MCA device drivers should identify the parent nexus driver implicitly using the **class** keyword. This removes the dependency on the name of the particular nexus driver involved since most drivers can operate device controllers attached to any of those buses. The ISA, EISA, and MCA nexus drivers all belong to class **sysbus**. See **driver.conf**(4) for further details of configuration file syntax.

All bus drivers of class **sysbus** recognize the following properties:

intr An arbitrary-length array where each element of the array consists of a pair of integers. Each array element describes a possible interrupt that the device might generate.

The first integer of each pair specifies the device's relative priority. This is the priority that this device's interrupt handler will receive relative to the interrupt handlers of other drivers. The priority is an integer from **1** to **16**. Generally, disks are assigned a priority of **5**, while mice and printers are lower, and serial communication devices are higher, typically **7**. **10** is reserved by the system and must not be used. Interrupts **11** and greater are high level interrupts and are generally not recommended (see **ddi_intr_hilevel**(9F)).

The second integer of each pair denotes the hardware interrupt request (IRQ) number.

The driver can refer to the elements of this array by index using **ddi_add_intr**(9F). The index into the array is passed as the *inumber* argument of **ddi_add_intr( )**.

Only devices that generate interrupts need to provide **intr** properties.

reg An arbitrary-length array where each element of the array consists of a 3-tuple of integers. Each array element describes a contiguous memory address range associated with the device on the bus.

The first integer of the tuple is reserved.

The driver can refer to the elements of this array by index, and construct kernel mappings to these addresses using **ddi_map_regs**(9F). The index into the array is passed as the *rnumber* argument of **ddi_map_regs( )**.

All **sysbus** device drivers must provide **reg** properties. The first two

integer elements of this property are used to construct the address part of the device name under **/devices**. If the device has memory-mapped addresses, the first integer should be **0** and the second integer should specify the physical address; if the device does not have memory-mapped addresses, the first integer should specify a unique identifier and the second integer should be **0**. A recommended unique identifier is the **ioaddr** value; that is, specify the I/O address in both the **ioaddr** field and the first integer of the **reg** field. The third integer of each 3-tuple specifies the size, in bytes, of the mappable region.

It is recommended that drivers for devices connected to the system bus recognize the following standard property names:

ioaddr An integer that describes the I/O port base address for this device.

dmachan An integer that specifies the DMA channel used by this device. Only devices that use a DMA channel need to provide **dmachan** properties.

EXAMPLES

Here are three sample entries from three different **driver.conf** files:

```
name="fdc" class="sysbus" intr=5,6 ioaddr=0x3f0 dmachan=2 reg=0x3f0,0,0;

name="logi" class="sysbus" intr=1,4 ioaddr=0x23c reg=0x23c,0,0;

name="sbpro" class="sysbus" ioaddr=0x220 intr=5,7 dmachan=1 reg=0x220,0,0;
```

SEE ALSO

driver.conf(4), **scsi**(4), **ddi_add_intr**(9F), **ddi_intr_hilevel**(9F), **ddi_map_regs**(9F), **ddi_prop_op**(9F)

Writing Device Drivers

NAME

syslog.conf – configuration file for syslogd system log daemon

SYNOPSIS

/etc/syslog.conf

DESCRIPTION

The file **/etc/syslog.conf** contains information used by the system log daemon, **syslogd**(1M), to forward a system message to appropriate log files and / or users. **syslogd** preprocesses this file through **m4**(1) to obtain the correct information for certain log files, defining **LOGHOST** if the address of "loghost" is the same as one of the addresses of the host that is running **syslogd**.

A configuration entry is composed of two TAB-separated fields:

"selector action"

The *selector* field contains a semicolon-separated list of priority specifications of the form:

*facility***.***level* [**;** *facility***.***level*]

where *facility* is a system facility, or comma-separated list of facilities, and *level* is an indication of the severity of the condition being logged. Recognized values for *facility* include:

user Messages generated by user processes. This is the default priority for messages from programs or facilities not listed in this file.

kern Messages generated by the kernel.

mail The mail system.

daemon System daemons, such as **in.ftpd**(1M)

auth The authorization system: **login**(1), **su**(1M), **getty**(1M), etc.

lpr The line printer spooling system: **lpr**(1B), **lpc**(1B), etc.

news Reserved for the USENET network news system.

uucp Reserved for the UUCP system; it does not currently use the **syslog** mechanism.

cron The **cron /**at facility; **crontab**(1), **at**(1), **cron**(1M), etc.

local0-7 Reserved for local use.

mark For timestamp messages produced internally by **syslogd**.

***** An asterisk indicates all facilities except for the **mark** facility.

Recognized values for *level* are (in descending order of severity):

emerg For panic conditions that would normally be broadcast to all users.

alert For conditions that should be corrected immediately, such as a corrupted system database.

crit For warnings about critical conditions, such as hard device errors.

err For other errors.

warning For warning messages.

notice For conditions that are not error conditions, but may require special handling.

info Informational messages.

debug For messages that are normally used only when debugging a program.

none Do not send messages from the indicated *facility* to the selected file. For example, a *selector* of

***.debug;mail.none**

will send all messages *except* mail messages to the selected file.

The *action* field indicates where to forward the message. Values for this field can have one of four forms:

- A filename, beginning with a leading slash, which indicates that messages specified by the *selector* are to be written to the specified file. The file will be opened in append mode.
- The name of a remote host, prefixed with an **@**, as with: **@***server*, which indicates that messages specified by the *selector* are to be forwarded to the **syslogd** on the named host. The hostname "loghost" is the hostname given to the machine that will log **syslogd** messages. Every machine is "loghost" by default. See **/etc/hosts**. It is also possible to specify one machine on a network to be "loghost" by making the appropriate host table entries. If the local machine is designated to be "loghost", then **syslogd** messages are written to the appropriate files. Otherwise, they are sent to the machine "loghost" on the network.
- A comma-separated list of usernames, which indicates that messages specified by the *selector* are to be written to the named users if they are logged in.
- An asterisk, which indicates that messages specified by the *selector* are to be written to all logged-in users.

Blank lines are ignored. Lines for which the first nonwhite character is a '**#**' are treated as comments.

EXAMPLES

With the following configuration file:

***.notice;mail.info**	**/var/log/notice**
***.crit**	**/var/log/critical**
kern,mark.debug	**/dev/console**
kern.err	**@server**
***.emerg**	*
***.alert**	**root,operator**
***.alert;auth.warning**	**/var/log/auth**

syslogd will log all mail system messages except **debug** messages and all **notice** (or higher) messages into a file named **/var/log/notice**. It logs all critical messages into **/var/log/critical**, and all kernel messages and 20-minute marks onto the system console.

Kernel messages of **err** (error) severity or higher are forwarded to the machine named *server*. Emergency messages are forwarded to all users. The users "root" and "operator" are informed of any **alert** messages. All messages from the authorization system of **warning** level or higher are logged in the file **/var/log/auth**.

FILES

/var/log/notice	log of all mail system messages (except **debug** messages) and all messages of **notice** level or higher.
/var/log/critical	log of all **critical** messages
/var/log/auth	log of all messages from the authorization system of **warning** level or higher

SEE ALSO

at(1), **crontab**(1), **logger**(1), **login**(1), **lp**(1), **m4**(1), **lpr**(1B), **cron**(1M), **getty**(1M), **in.ftpd**(1M), **su**(1M), **syslogd**(1M), **syslog**(3)

NAME

system – system configuration information file

DESCRIPTION

The **system** file is used for customizing the operation of the operating system kernel. The recommended procedure is to preserve the original **system** file before modifying it.

The **system** file contains commands which are read by the kernel during initialization and used to customize the operation of your system. These commands are useful for modifying the system's treatment of its loadable kernel modules.

The syntax of the **system** file consists of a list of keyword / value pairs which are recognized by the system as valid commands. Comment lines must begin with an asterisk ('*') and end with a newline character. All commands are case-insensitive except where noted. A command line can be no more than **80** characters in length.

Commands that modify the system's operation with respect to loadable kernel modules require you to specify the module type by listing the module's namespace. The following namespaces are currently supported:

drv Modules in this namespace are device drivers.

exec Modules in this namespace are execution format modules. The following **exec** modules are currently provided by SunSoft:

SPARC system: **aoutexec**, **elfexec**, **intpexec**

x86 system: **coffexec**, **elfexec**, **intpexec**

fs These modules are filesystems.

sched These modules implement a process scheduling algorithm.

strmod These modules are STREAMS modules.

sys These modules implement loadable system-call modules.

misc These modules do not fit into any of the above categories, so are considered "miscellaneous" modules.

Below is a description of each of the supported commands:

exclude: <*namespace*> / <*modulename*>
Do not allow the listed loadable kernel module to be loaded. **exclude** commands are cumulative; the list of modules to **exclude** is created by combining every **exclude** entry in the **system** file.

include: <*namespace*> / <*modulename*>
Include the listed loadable kernel module. This is the system's default, so using **include** does not modify the system's operation. **include** commands are cumulative.

forceload: <*namespace*> / <*modulename*>
Force this kernel module to be loaded during kernel initialization. The default action is to automatically load the kernel module when its services are first accessed. **forceload** commands are cumulative.

rootdev: <*device name*>
Set the root device to the listed value instead of using the default root device as supplied by the boot program.

rootfs: <*root filesystem type*>
Set the root filesystem type to the listed value.

moddir: <*first module path*>[[:,]<*second ...*>]...]
Set the search path for loadable kernel modules. This command operates very much like the **PATH** shell variable. Multiple directories to search can be listed together, delimited either by blank spaces or colons.

set [<*module*>*:*]<*symbol*> {=, |, &} [~][-]<*value*>
Set an integer or character pointer in the kernel or in the selected kernel module to a new value. This command is used to change kernel and module parameters and thus modify the operation of your system. Assignment operations are not cumulative, whereas bitwise AND and OR operations are cumulative.

Operations that are supported for modifying integer variables are: simple assignment, inclusive bitwise OR, bitwise AND, one's complement, and negation. Variables in a specific loadable module can be targeted for modification by specifying the variable name prefixed with the kernel module name and a colon (:) separator. Values can be specified as hexadecimal (0x10), Octal (046), or Decimal (5).

The only operation supported for modifying character pointers is simple assignment. Static string data such as character arrays cannot be modified using the **set** command. Use care and ensure that the variable you are modifying is in fact a character pointer. The **set** command is very powerful, and will likely cause problems if used carelessly. The entire command, including the quoted string, cannot exceed **80** characters. The following escape sequences are supported within the quoted string:

\n (newline)
\t (tab)
\b (backspace)

EXAMPLES

The following is a sample **system** file.

```
* Force the ELF exec kernel module to be loaded during kernel
* initialization. Execution type modules are in the exec namespace.
forceload: exec/elfexec
```

```
* Change the root device to /sbus@1,f8000000/esp@0,800000/sd@3,0:a.
* You can derive root device names from /devices.
* Root device names must be the fully expanded Open Boot Prom
* device name.  This command is platform and configuration specific.
* This example uses the first partition (a) of the SCSI disk at
* SCSI target 3 on the esp host adapter in slot 0 (on board)
* of the SBus of the machine.
* Adapter unit-address 3,0 at sbus unit-address 0,800000.
rootdev: /sbus@1,f8000000/esp@0,800000/sd@3,0:a

* Set the filesystem type of the root to ufs. Note that
* the equal sign can be used instead of the colon.
rootfs:ufs

* Set the search path for kernel modules to look first in
* /usr/phil/mod_test for modules, then in /kernel/modules (the
* default) if not found. Useful for testing new modules.
* Note that you can delimit your module pathnames using
* colons instead of spaces: moddir:/newmodules:/kernel/modules
moddir:/usr/phil/mod_test /kernel/modules.

* Set the integer variable "maxusers" in the kernel to 16. This is a
* useful tuning parameter.
set maxusers = 16

* Turn on debugging messages in the modules mydriver. This is useful
* during driver development.
set mydriver:debug = 1

* Bitwise AND the kernel variable "moddebug" with the
* one's complement of the hex value 0x880, and set
* "moddebug" to this new value.
set moddebug & ~0x880

* Demonstrate the cumulative effect of the SET
* bitwise AND/OR operations by further modifying "moddebug"
* by ORing it with 0x40.
set moddebug | 0x40
```

WARNINGS

system file lines must be fewer than **80** characters in length.

Use care when modifying the **system** file; it modifies the operation of the kernel. If you preserved the original **system** file, you can use the **boot -a** option and supply the path to the original file, allowing the system to boot correctly.

NOTES **/etc/system** is only read once: at boot time.

NAME

term – format of compiled term file

SYNOPSIS

/usr/share/lib/terminfo/?/*

DESCRIPTION

Compiled **terminfo**(4) descriptions are placed under the directory **/usr/share/lib/terminfo**. In order to avoid a linear search of a huge system directory, a two-level scheme is used: **/usr/share/lib/terminfo/***c*/*name* where *name* is the name of the terminal, and *c* is the first character of *name*. Thus, **att4425** can be found in the file **/usr/share/lib/terminfo/a/att4425**. Synonyms for the same terminal are implemented by multiple links to the same compiled file.

The format has been chosen so that it is the same on all hardware. An 8-bit byte is assumed, but no assumptions about byte ordering or sign extension are made. Thus, these binary **terminfo** files can be transported to other hardware with 8-bit bytes.

Short integers are stored in two 8-bit bytes. The first byte contains the least significant 8 bits of the value, and the second byte contains the most significant 8 bits. (Thus, the value represented is 256**second*+*first*.) The value **–1** is represented by **0377,0377**, and the value **–2** is represented by **0376,0377**; other negative values are illegal. The **–1** generally means that a capability is missing from this terminal. The **–2** means that the capability has been cancelled in the **terminfo** source and also is to be considered missing.

The compiled file is created from the source file descriptions of the terminals (see the **–I** option of **infocmp**) by using the **terminfo** compiler, **tic**, and read by the routine **setupterm** (see **curses**(3X)). The file is divided into six parts in the following order: the header, terminal names, boolean flags, numbers, strings, and string table.

The header section begins the file. This section contains six short integers in the format described below. These integers are (1) the magic number (octal **0432**); (2) the size, in bytes, of the names section; (3) the number of bytes in the boolean section; (4) the number of short integers in the numbers section; (5) the number of offsets (short integers) in the strings section; (6) the size, in bytes, of the string table.

The terminal names section comes next. It contains the first line of the **terminfo** description, listing the various names for the terminal, separated by the bar (|) character (see **term**(5)). The section is terminated with an ASCII NUL character.

The boolean flags have one byte for each flag. This byte is either **0** or **1** as the flag is present or absent. The value of **2** means that the flag has been cancelled. The capabilities are in the same order as the file <**term.h**>.

Between the boolean section and the number section, a null byte is inserted, if necessary, to ensure that the number section begins on an even byte offset. All short integers are aligned on a short word boundary.

The numbers section is similar to the boolean flags section. Each capability takes up two bytes, and is stored as a short integer. If the value represented is **–1** or **–2**, the capability is taken to be missing.

The strings section is also similar. Each capability is stored as a short integer, in the format above. A value of **–1** or **–2** means the capability is missing. Otherwise, the value is taken as an offset from the beginning of the string table. Special characters in ^X or \c notation are stored in their interpreted form, not the printing representation. Padding information ($<nn>) and parameter information (%x) are stored intact in uninterpreted form.

The final section is the string table. It contains all the values of string capabilities referenced in the string section. Each string is null terminated.

Note that it is possible for **setupterm** to expect a different set of capabilities than are actually present in the file. Either the database may have been updated since **setupterm** has been recompiled (resulting in extra unrecognized entries in the file) or the program may have been recompiled more recently than the database was updated (resulting in missing entries). The routine **setupterm** must be prepared for both possibilities—this is why the numbers and sizes are included. Also, new capabilities must always be added at the end of the lists of boolean, number, and string capabilities.

As an example, here is terminal information on the AT&T Model 37 KSR terminal as output by the **infocmp –I tty37** command:

```
37|tty37|AT&T model 37 teletype,
    hc, os, xon,
    bel=^G, cr=\r, cub1=\b, cud1=\n, cuu1=\E7, hd=\E9,
    hu=\E8, ind=\n,
```

And here is an octal dump of the **term** file, produced by the **od -c /usr/share/lib/terminfo/t/tty37** command:

```
0000000 032 001    \0 032  \0 013  \0 021 001  3  \0  3  7  |  t
0000020  t  y  3  7  |  A  T  &  T    m  o  d  e  l
0000040  3  7    t  e  l  e  t  y  p  e \0 \0 \0 \0 \0
0000060 \0 \0 \0 001 \0 \0 \0 \0 \0 \0 \0 001 \0 \0 \0 \0
0000100 001 \0 \0 \0 \0 \0 377 377 377 377 377 377 377 377 377 377
0000120 377 377 377 377 377 377 377 377 377 377 377 377 377 377  &  \0
0000140    \0 377 377 377 377 377 377 377 377 377 377 377 377 377 377
0000160 377 377  "  \0 377 377 377 377  (  \0 377 377 377 377 377 377
0000200 377 377  0  \0 377 377 377 377 377 377 377 377  -  \0 377 377
0000220 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
0000520 377 377 377 377 377 377 377 377 377 377 377 377 377 377  $  \0
0000540 377 377 377 377 377 377 377 377 377 377 377 377 377 377  *  \0
0000560 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
```

```
0001160 377 377 377 377 377 377 377 377 377 377 377 377 377 377  3  7
0001200  |  t  t  y  3  7  |  A  T  &  T     m  o  d  e
0001220  l     3  7     t  e  l  e  t  y  p  e \0 \r \0
0001240 \n \0 \n \0 007 \0 \b \0 033  8 \0 033  9 \0 033  7
0001260 \0 \0
0001261
```

Some limitations: total compiled entries cannot exceed 4096 bytes; all entries in the name field cannot exceed 128 bytes.

FILES

/usr/share/lib/terminfo/?/*	compiled terminal description database
/usr/include/term.h	**terminfo** header

SEE ALSO

infocmp(1M), **curses**(3X), **terminfo**(4), **term**(5)

NAME

terminfo – terminal capability database

SYNOPSIS

/usr/share/lib/terminfo/?/*

DESCRIPTION

terminfo is a database produced by **tic** that describes the capabilities of devices such as terminals and printers. Devices are described in **terminfo** source files by specifying a set of capabilities, by quantifying certain aspects of the device, and by specifying character sequences that effect particular results. This database is often used by screen oriented applications such as **vi** and **curses** programs, as well as by some system commands such as **ls** and **more**. This usage allows them to work with a variety of devices without changes to the programs.

terminfo source files consist of one or more device descriptions. Each description consists of a header (beginning in column 1) and one or more lines that list the features for that particular device. Every line in a **terminfo** source file must end in a comma (**,**). Every line in a **terminfo** source file except the header must be indented with one or more white spaces (either spaces or tabs).

Entries in **terminfo** source files consist of a number of comma-separated fields. White space after each comma is ignored. Embedded commas must be escaped by using a backslash. The following example shows the format of a **terminfo** source file.

$alias sub 1$ | $alias sub 2$ | ... | $alias sub n$ | *longname*,
<*white space*> **am, lines #24,**
<*white space*> **home=\Eeh,**

The first line, commonly referred to as the header line, must begin in column one and must contain at least two aliases separated by vertical bars. The last field in the header line must be the long name of the device and it may contain any string. Alias names must be unique in the **terminfo** database and they must conform to system file naming conventions (see **tic**(1M)); they cannot, for example, contain white space or slashes.

Every device must be assigned a name, such as "vt100". Device names (except the long name) should be chosen using the following conventions. The name should not contain hyphens because hyphens are reserved for use when adding suffixes that indicate special modes.

These special modes may be modes that the hardware can be in, or user preferences. To assign a special mode to a particular device, append a suffix consisting of a hyphen and an indicator of the mode to the device name. For example, the **-w** suffix means "wide mode"; when specified, it allows for a width of 132 columns instead of the standard 80 columns.

Therefore, if you want to use a "vt100" device set to wide mode, name the device "vt100-w." Use the following suffixes where possible.

Suffix	Meaning	Example
-w	Wide mode (more than 80 columns)	**5410-w**
-am	With auto. margins (usually default)	**vt100-am**
-nam	Without automatic margins	**vt100-nam**
-n	Number of lines on the screen	**2300-40**
-na	No arrow keys (leave them in local)	**c100-na**
*-n*p	Number of pages of memory	**c100-4p**
-rv	Reverse video	**4415-rv**

The **terminfo** reference manual page is organized in two sections:

DEVICE CAPABILITIES and PRINTER CAPABILITIES.

PART 1: DEVICE CAPABILITIES

Capabilities in **terminfo** are of three types: Boolean capabilities (which show that a device has or does not have a particular feature), numeric capabilities (which quantify particular features of a device), and string capabilities (which provide sequences that can be used to perform particular operations on devices).

In the following table, a **Variable** is the name by which a **C** programmer accesses a capability (at the **terminfo** level). A **Capname** is the short name for a capability specified in the **terminfo** source file. It is used by a person updating the source file and by the **tput** command. A **Termcap Code** is a two-letter sequence that corresponds to the **termcap** capability name. (Note that **termcap** is no longer supported.)

Capability names have no real length limit, but an informal limit of five characters has been adopted to keep them short. Whenever possible, capability names are chosen to be the same as or similar to those specified by the ANSI X3.64-1979 standard. Semantics are also intended to match those of the ANSI standard.

All string capabilities listed below may have padding specified, with the exception of those used for input. Input capabilities, listed under the **Strings** section in the following tables, have names beginning with **key_**. The **#i** symbol in the description field of the following tables refers to the *i*th parameter.

Booleans

Variable	Cap-name	Termcap Code	Description
auto_left_margin	**bw**	**bw**	**cub1** wraps from column 0 to last column
auto_right_margin	**am**	**am**	Terminal has automatic margins
back_color_erase	**bce**	**be**	Screen erased with background color
can_change	**ccc**	**cc**	Terminal can re-define existing color
ceol_standout_glitch	**xhp**	**xs**	Standout not erased by overwriting (hp)
col_addr_glitch	**xhpa**	**YA**	Only positive motion for **hpa**/**mhpa** caps
cpi_changes_res	**cpix**	**YF**	Changing character pitch changes resolution
cr_cancels_micro_mode	**crxm**	**YB**	Using **cr** turns off micro mode

eat_newline_glitch	**xenl**	**xn**	Newline ignored after 80 columns (Concept)
erase_overstrike	**eo**	**eo**	Can erase overstrikes with a blank
generic_type	**gn**	**gn**	Generic line type (for example, dialup, switch)
hard_copy	**hc**	**hc**	Hardcopy terminal
hard_cursor	**chts**	**HC**	Cursor is hard to see
has_meta_key	**km**	**km**	Has a meta key (shift, sets parity bit)
has_print_wheel	**daisy**	**YC**	Printer needs operator to change character set
has_status_line	**hs**	**hs**	Has extra "status line"
hue_lightness_saturation	**hls**	**hl**	Terminal uses only HLS color notation (Tektronix)
insert_null_glitch	**in**	**in**	Insert mode distinguishes nulls
lpi_changes_res	**lpix**	**YG**	Changing line pitch changes resolution
memory_above	**da**	**da**	Display may be retained above the screen
memory_below	**db**	**db**	Display may be retained below the screen
move_insert_mode	**mir**	**mi**	Safe to move while in insert mode
move_standout_mode	**msgr**	**ms**	Safe to move in standout modes
needs_xon_xoff	**nxon**	**nx**	Padding won't work, xon/xoff required
no_esc_ctlc	**xsb**	**xb**	Beehive (f1=escape, f2=ctrl C)
non_rev_rmcup	**nrrmc**	**NR**	**smcup** does not reverse **rmcup**
no_pad_char	**npc**	**NP**	Pad character doesn't exist
over_strike	**os**	**os**	Terminal overstrikes on hard-copy terminal
prtr_silent	**mc5i**	**5i**	Printer won't echo on screen
row_addr_glitch	**xvpa**	**YD**	Only positive motion for **vpa**/**mvpa** caps
semi_auto_right_margin	**sam**	**YE**	Printing in last column causes **cr**
status_line_esc_ok	**eslok**	**es**	Escape can be used on the status line
dest_tabs_magic_smso	**xt**	**xt**	Destructive tabs, magic **smso** char (t1061)
tilde_glitch	**hz**	**hz**	Hazeltine; can't print tilde (~)
transparent_underline	**ul**	**ul**	Underline character overstrikes
xon_xoff	**xon**	**xo**	Terminal uses xon/xoff handshaking

Numbers

Variable	Cap-name	Termcap Code	Description
buffer_capacity	**bufsz**	**Ya**	Number of bytes buffered before printing
columns	**cols**	**co**	Number of columns in a line
dot_vert_spacing	**spinv**	**Yb**	Spacing of pins vertically in pins per inch
dot_horz_spacing	**spinh**	**Yc**	Spacing of dots horizontally in dots per inch
init_tabs	**it**	**it**	Tabs initially every # spaces
label_height	**lh**	**lh**	Number of rows in each label
label_width	**lw**	**lw**	Number of columns in each label
lines	**lines**	**li**	Number of lines on a screen or a page
lines_of_memory	**lm**	**lm**	Lines of memory if > **lines**; **0** means varies

magic_cookie_glitch	**xmc**	**sg**	Number of blank characters left by **smso** or **rmso**
max_colors	**colors**	**Co**	Maximum number of colors on the screen
max_micro_address	**maddr**	**Yd**	Maximum value in **micro_..._address**
max_micro_jump	**mjump**	**Ye**	Maximum value in **parm_..._micro**
max_pairs	**pairs**	**pa**	Maximum number of color-pairs on the screen
micro_col_size	**mcs**	**Yf**	Character step size when in micro mode
micro_line_size	**mls**	**Yg**	Line step size when in micro mode
no_color_video	**ncv**	**NC**	Video attributes that can't be used with colors
number_of_pins	**npins**	**Yh**	Number of pins in print-head
num_labels	**nlab**	**Nl**	Number of labels on screen (start at 1)
output_res_char	**orc**	**Yi**	Horizontal resolution in units per character
output_res_line	**orl**	**Yj**	Vertical resolution in units per line
output_res_horz_inch	**orhi**	**Yk**	Horizontal resolution in units per inch
output_res_vert_inch	**orvi**	**Yl**	Vertical resolution in units per inch
padding_baud_rate	**pb**	**pb**	Lowest baud rate where padding needed
virtual_terminal	**vt**	**vt**	Virtual terminal number (system)
wide_char_size	**widcs**	**Yn**	Character step size when in double wide mode
width_status_line	**wsl**	**ws**	Number of columns in status line

Strings

Variable	Cap-name	Termcap Code	Description
acs_chars	**acsc**	**ac**	Graphic charset pairs aAbBcC
alt_scancode_esc	**scesca**	**S8**	Alternate escape for scancode emulation (default is for vt100)
back_tab	**cbt**	**bt**	Back tab
bell	**bel**	**bl**	Audible signal (bell)
bit_image_repeat	**birep**	**Zy**	Repeat bit-image cell #1 #2 times (use tparm)
bit_image_newline	**binel**	**Zz**	Move to next row of the bit image (use tparm)
bit_image_carriage_return	**bicr**	**Yv**	Move to beginning of same row (use tparm)
carriage_return	**cr**	**cr**	Carriage return
change_char_pitch	**cpi**	**ZA**	Change number of characters per inch
change_line_pitch	**lpi**	**ZB**	Change number of lines per inch
change_res_horz	**chr**	**ZC**	Change horizontal resolution
change_res_vert	**cvr**	**ZD**	Change vertical resolution
change_scroll_region	**csr**	**cs**	Change to lines #1 through #2 (vt100)
char_padding	**rmp**	**rP**	Like **ip** but when in replace mode
char_set_names	**csnm**	**Zy**	List of character set names
clear_all_tabs	**tbc**	**ct**	Clear all tab stops
clear_margins	**mgc**	**MC**	Clear all margins (top, bottom, and sides)

clear_screen	**clear**	**cl**	Clear screen and home cursor
clr_bol	**el1**	**cb**	Clear to beginning of line, inclusive
clr_eol	**el**	**ce**	Clear to end of line
clr_eos	**ed**	**cd**	Clear to end of display
code_set_init	**csin**	**ci**	Init sequence for multiple codesets
color_names	**colornm**	**Yw**	Give name for color #1
column_address	**hpa**	**ch**	Horizontal position absolute
command_character	**cmdch**	**CC**	Terminal settable cmd character in prototype
cursor_address	**cup**	**cm**	Move to row #1 col #2
cursor_down	**cud1**	**do**	Down one line
cursor_home	**home**	**ho**	Home cursor (if no **cup**)
cursor_invisible	**civis**	**vi**	Make cursor invisible
cursor_left	**cub1**	**le**	Move left one space.
cursor_mem_address	**mrcup**	**CM**	Memory relative cursor addressing
cursor_normal	**cnorm**	**ve**	Make cursor appear normal (undo **vs/vi**)
cursor_right	**cuf1**	**nd**	Non-destructive space (cursor or carriage right)
cursor_to_ll	**ll**	**ll**	Last line, first column (if no **cup**)
cursor_up	**cuu1**	**up**	Upline (cursor up)
cursor_visible	**cvvis**	**vs**	Make cursor very visible
define_bit_image_region	**defbi**	**Yx**	Define rectangular bit-image region (use tparm)
define_char	**defc**	**ZE**	Define a character in a character set †
delete_character	**dch1**	**dc**	Delete character
delete_line	**dl1**	**dl**	Delete line
device_type	**devt**	**dv**	Indicate language/codeset support
dis_status_line	**dsl**	**ds**	Disable status line
display_pc_char	**dispc**	**S1**	Display PC character
down_half_line	**hd**	**hd**	Half-line down (forward 1/2 linefeed)
ena_acs	**enacs**	**eA**	Enable alternate character set
end_bit_image_region	**endbi**	**Yy**	End a bit-image region (use tparm)
enter_alt_charset_mode	**smacs**	**as**	Start alternate character set
enter_am_mode	**smam**	**SA**	Turn on automatic margins
enter_blink_mode	**blink**	**mb**	Turn on blinking
enter_bold_mode	**bold**	**md**	Turn on bold (extra bright) mode
enter_ca_mode	**smcup**	**ti**	String to begin programs that use **cup**
enter_delete_mode	**smdc**	**dm**	Delete mode (enter)
enter_dim_mode	**dim**	**mh**	Turn on half-bright mode
enter_doublewide_mode	**swidm**	**ZF**	Enable double wide printing
enter_draft_quality	**sdrfq**	**ZG**	Set draft quality print
enter_insert_mode	**smir**	**im**	Insert mode (enter)
enter_italics_mode	**sitm**	**ZH**	Enable italics
enter_leftward_mode	**slm**	**ZI**	Enable leftward carriage motion

enter_micro_mode	**smicm**	**ZJ**	Enable micro motion capabilities
enter_near_letter_quality	**snlq**	**ZK**	Set near-letter quality print
enter_normal_quality	**snrmq**	**ZL**	Set normal quality print
enter_pc_charset_mode	**smpch**	**S2**	Enter PC character display mode
enter_protected_mode	**prot**	**mp**	Turn on protected mode
enter_reverse_mode	**rev**	**mr**	Turn on reverse video mode
enter_scancode_mode	**smsc**	**S4**	Enter PC scancode mode
enter_secure_mode	**invis**	**mk**	Turn on blank mode (characters invisible)
enter_shadow_mode	**sshm**	**ZM**	Enable shadow printing
enter_standout_mode	**smso**	**so**	Begin standout mode
enter_subscript_mode	**ssubm**	**ZN**	Enable subscript printing
enter_superscript_mode	**ssupm**	**ZO**	Enable superscript printing
enter_underline_mode	**smul**	**us**	Start underscore mode
enter_upward_mode	**sum**	**ZP**	Enable upward carriage motion
enter_xon_mode	**smxon**	**SX**	Turn on xon / xoff handshaking
erase_chars	**ech**	**ec**	Erase #1 characters
exit_alt_charset_mode	**rmacs**	**ae**	End alternate character set
exit_am_mode	**rmam**	**RA**	Turn off automatic margins
exit_attribute_mode	**sgr0**	**me**	Turn off all attributes
exit_ca_mode	**rmcup**	**te**	String to end programs that use **cup**
exit_delete_mode	**rmdc**	**ed**	End delete mode
exit_doublewide_mode	**rwidm**	**ZQ**	Disable double wide printing
exit_insert_mode	**rmir**	**ei**	End insert mode
exit_italics_mode	**ritm**	**ZR**	Disable italics
exit_leftward_mode	**rlm**	**ZS**	Enable rightward (normal) carriage motion
exit_micro_mode	**rmicm**	**ZT**	Disable micro motion capabilities
exit_pc_charset_mode	**rmpch**	**S3**	Disable PC character display mode
exit_scancode_mode	**rmsc**	**S5**	Disable PC scancode mode
exit_shadow_mode	**rshm**	**ZU**	Disable shadow printing
exit_standout_mode	**rmso**	**se**	End standout mode
exit_subscript_mode	**rsubm**	**ZV**	Disable subscript printing
exit_superscript_mode	**rsupm**	**ZW**	Disable superscript printing
exit_underline_mode	**rmul**	**ue**	End underscore mode
exit_upward_mode	**rum**	**ZX**	Enable downward (normal) carriage motion
exit_xon_mode	**rmxon**	**RX**	Turn off xon / xoff handshaking
flash_screen	**flash**	**vb**	Visible bell (may not move cursor)
form_feed	**ff**	**ff**	Hardcopy terminal page eject
from_status_line	**fsl**	**fs**	Return from status line
init_1string	**is1**	**i1**	Terminal or printer initialization string
init_2string	**is2**	**is**	Terminal or printer initialization string
init_3string	**is3**	**i3**	Terminal or printer initialization string
init_file	**if**	**if**	Name of initialization file

init_prog	**iprog**	**iP**	Path name of program for initialization
initialize_color	**initc**	**Ic**	Initialize the definition of color
initialize_pair	**initp**	**Ip**	Initialize color-pair
insert_character	**ich1**	**ic**	Insert character
insert_line	**il1**	**al**	Add new blank line
insert_padding	**ip**	**ip**	Insert pad after character inserted

The "**key_**" strings are sent by specific keys. The "**key_**" descriptions include the macro, defined in **<curses.h>**, for the code returned by the **curses** routine **getch** when the key is pressed (see **curs_getch**(3X)).

Variable	Cap-name	Termcap Code	Description
key_a1	**ka1**	**K1**	**KEY_A1**, upper left of keypad
key_a3	**ka3**	**K3**	**KEY_A3**, upper right of keypad
key_b2	**kb2**	**K2**	**KEY_B2**, center of keypad
key_backspace	**kbs**	**kb**	**KEY_BACKSPACE**, sent by backspace key
key_beg	**kbeg**	**@1**	**KEY_BEG**, sent by beg(inning) key
key_btab	**kcbt**	**kB**	**KEY_BTAB**, sent by back-tab key
key_c1	**kc1**	**K4**	**KEY_C1**, lower left of keypad
key_c3	**kc3**	**K5**	**KEY_C3**, lower right of keypad
key_cancel	**kcan**	**@2**	**KEY_CANCEL**, sent by cancel key
key_catab	**ktbc**	**ka**	**KEY_CATAB**, sent by clear-all-tabs key
key_clear	**kclr**	**kC**	**KEY_CLEAR**, sent by clear-screen or erase key
key_close	**kclo**	**@3**	**KEY_CLOSE**, sent by close key
key_command	**kcmd**	**@4**	**KEY_COMMAND**, sent by cmd (command) key
key_copy	**kcpy**	**@5**	**KEY_COPY**, sent by copy key
key_create	**kcrt**	**@6**	**KEY_CREATE**, sent by create key
key_ctab	**kctab**	**kt**	**KEY_CTAB**, sent by clear-tab key
key_dc	**kdch1**	**kD**	**KEY_DC**, sent by delete-character key
key_dl	**kdl1**	**kL**	**KEY_DL**, sent by delete-line key
key_down	**kcud1**	**kd**	**KEY_DOWN**, sent by terminal down-arrow key
key_eic	**krmir**	**kM**	**KEY_EIC**, sent by **rmir** or **smir** in insert mode
key_end	**kend**	**@7**	**KEY_END**, sent by end key
key_enter	**kent**	**@8**	**KEY_ENTER**, sent by enter/send key
key_eol	**kel**	**kE**	**KEY_EOL**, sent by clear-to-end-of-line key
key_eos	**ked**	**kS**	**KEY_EOS**, sent by clear-to-end-of-screen key
key_exit	**kext**	**@9**	**KEY_EXIT**, sent by exit key
key_f0	**kf0**	**k0**	**KEY_F(0)**, sent by function key f0

key_f1	**kf1**	**k1**	**KEY_F(1)**, sent by function key f1
key_f2	**kf2**	**k2**	**KEY_F(2)**, sent by function key f2
key_f3	**kf3**	**k3**	**KEY_F(3)**, sent by function key f3
key_fB	**kfB**	**k4**	**KEY_F(4)**, sent by function key fB
key_f5	**kf5**	**k5**	**KEY_F(5)**, sent by function key f5
key_f6	**kf6**	**k6**	**KEY_F(6)**, sent by function key f6
key_f7	**kf7**	**k7**	**KEY_F(7)**, sent by function key f7
key_f8	**kf8**	**k8**	**KEY_F(8)**, sent by function key f8
key_f9	**kf9**	**k9**	**KEY_F(9)**, sent by function key f9
key_f10	**kf10**	**k;**	**KEY_F(10)**, sent by function key f10
key_f11	**kf11**	**F1**	**KEY_F(11)**, sent by function key f11
key_f12	**kf12**	**F2**	**KEY_F(12)**, sent by function key f12
key_f13	**kf13**	**F3**	**KEY_F(13)**, sent by function key f13
key_f14	**kf14**	**F4**	**KEY_F(14)**, sent by function key f14
key_f15	**kf15**	**F5**	**KEY_F(15)**, sent by function key f15
key_f16	**kf16**	**F6**	**KEY_F(16)**, sent by function key f16
key_f17	**kf17**	**F7**	**KEY_F(17)**, sent by function key f17
key_f18	**kf18**	**F8**	**KEY_F(18)**, sent by function key f18
key_f19	**kf19**	**F9**	**KEY_F(19)**, sent by function key f19
key_f20	**kf20**	**FA**	**KEY_F(20)**, sent by function key f20
key_f21	**kf21**	**FB**	**KEY_F(21)**, sent by function key f21
key_f22	**kf22**	**FC**	**KEY_F(22)**, sent by function key f22
key_f23	**kf23**	**FD**	**KEY_F(23)**, sent by function key f23
key_f24	**kf24**	**FE**	**KEY_F(24)**, sent by function key f24
key_f25	**kf25**	**FF**	**KEY_F(25)**, sent by function key f25
key_f26	**kf26**	**FG**	**KEY_F(26)**, sent by function key f26
key_f27	**kf27**	**FH**	**KEY_F(27)**, sent by function key f27
key_f28	**kf28**	**FI**	**KEY_F(28)**, sent by function key f28
key_f29	**kf29**	**FJ**	**KEY_F(29)**, sent by function key f29
key_f30	**kf30**	**FK**	**KEY_F(30)**, sent by function key f30
key_f31	**kf31**	**FL**	**KEY_F(31)**, sent by function key f31
key_f32	**kf32**	**FM**	**KEY_F(32)**, sent by function key f32
key_f33	**kf33**	**FN**	**KEY_F(13)**, sent by function key f13
key_f34	**kf34**	**FO**	**KEY_F(34)**, sent by function key f34
key_f35	**kf35**	**FP**	**KEY_F(35)**, sent by function key f35
key_f36	**kf36**	**FQ**	**KEY_F(36)**, sent by function key f36
key_f37	**kf37**	**FR**	**KEY_F(37)**, sent by function key f37
key_f38	**kf38**	**FS**	**KEY_F(38)**, sent by function key f38
key_f39	**kf39**	**FT**	**KEY_F(39)**, sent by function key f39
key_fB0	**kfB0**	**FU**	**KEY_F(40)**, sent by function key fB0
key_fB1	**kfB1**	**FV**	**KEY_F(41)**, sent by function key fB1
key_fB2	**kfB2**	**FW**	**KEY_F(42)**, sent by function key fB2
key_fB3	**kfB3**	**FX**	**KEY_F(43)**, sent by function key fB3
key_fB4	**kfB4**	**FY**	**KEY_F(44)**, sent by function key fB4
key_fB5	**kfB5**	**FZ**	**KEY_F(45)**, sent by function key fB5

key_fB6	**kfB6**	**Fa**	**KEY_F(46)**, sent by function key fB6
key_fB7	**kfB7**	**Fb**	**KEY_F(47)**, sent by function key fB7
key_fB8	**kfB8**	**Fc**	**KEY_F(48)**, sent by function key fB8
key_fB9	**kfB9**	**Fd**	**KEY_F(49)**, sent by function key fB9
key_f50	**kf50**	**Fe**	**KEY_F(50)**, sent by function key f50
key_f51	**kf51**	**Ff**	**KEY_F(51)**, sent by function key f51
key_f52	**kf52**	**Fg**	**KEY_F(52)**, sent by function key f52
key_f53	**kf53**	**Fh**	**KEY_F(53)**, sent by function key f53
key_f54	**kf54**	**Fi**	**KEY_F(54)**, sent by function key f54
key_f55	**kf55**	**Fj**	**KEY_F(55)**, sent by function key f55
key_f56	**kf56**	**Fk**	**KEY_F(56)**, sent by function key f56
key_f57	**kf57**	**Fl**	**KEY_F(57)**, sent by function key f57
key_f58	**kf58**	**Fm**	**KEY_F(58)**, sent by function key f58
key_f59	**kf59**	**Fn**	**KEY_F(59)**, sent by function key f59
key_f60	**kf60**	**Fo**	**KEY_F(60)**, sent by function key f60
key_f61	**kf61**	**Fp**	**KEY_F(61)**, sent by function key f61
key_f62	**kf62**	**Fq**	**KEY_F(62)**, sent by function key f62
key_f63	**kf63**	**Fr**	**KEY_F(63)**, sent by function key f63
key_find	**kfnd**	**@0**	**KEY_FIND**, sent by find key
key_help	**khlp**	**%1**	**KEY_HELP**, sent by help key
key_home	**khome**	**kh**	**KEY_HOME**, sent by home key
key_ic	**kich1**	**kI**	**KEY_IC**, sent by ins-char/enter ins-mode key
key_il	**kil1**	**kA**	**KEY_IL**, sent by insert-line key
key_left	**kcub1**	**kl**	**KEY_LEFT**, sent by terminal left-arrow key
key_ll	**kll**	**kH**	**KEY_LL**, sent by home-down key
key_mark	**kmrk**	**%2**	**KEY_MARK**, sent by mark key
key_message	**kmsg**	**%3**	**KEY_MESSAGE**, sent by message key
key_move	**kmov**	**%4**	**KEY_MOVE**, sent by move key
key_next	**knxt**	**%5**	**KEY_NEXT**, sent by next-object key
key_npage	**knp**	**kN**	**KEY_NPAGE**, sent by next-page key
key_open	**kopn**	**%6**	**KEY_OPEN**, sent by open key
key_options	**kopt**	**%7**	**KEY_OPTIONS**, sent by options key
key_ppage	**kpp**	**kP**	**KEY_PPAGE**, sent by previous-page key
key_previous	**kprv**	**%8**	**KEY_PREVIOUS**, sent by previous-object key
key_print	**kprt**	**%9**	**KEY_PRINT**, sent by print or copy key
key_redo	**krdo**	**%0**	**KEY_REDO**, sent by redo key
key_reference	**kref**	**&1**	**KEY_REFERENCE**, sent by reference key
key_refresh	**krfr**	**&2**	**KEY_REFRESH**, sent by refresh key
key_replace	**krpl**	**&3**	**KEY_REPLACE**, sent by replace key
key_restart	**krst**	**&4**	**KEY_RESTART**, sent by restart key
key_resume	**kres**	**&5**	**KEY_RESUME**, sent by resume key

key_right	**kcuf1**	**kr**	**KEY_RIGHT**, sent by terminal right-arrow key
key_save	**ksav**	**&6**	**KEY_SAVE**, sent by save key
key_sbeg	**kBEG**	**&9**	**KEY_SBEG**, sent by shifted beginning key
key_scancel	**kCAN**	**&0**	**KEY_SCANCEL**, sent by shifted cancel key
key_scommand	**kCMD**	**∗1**	**KEY_SCOMMAND**, sent by shifted command key
key_scopy	**kCPY**	**∗2**	**KEY_SCOPY**, sent by shifted copy key
key_screate	**kCRT**	**∗3**	**KEY_SCREATE**, sent by shifted create key
key_sdc	**kDC**	**∗4**	**KEY_SDC**, sent by shifted delete-char key
key_sdl	**kDL**	**∗5**	**KEY_SDL**, sent by shifted delete-line key
key_select	**kslt**	**∗6**	**KEY_SELECT**, sent by select key
key_send	**kEND**	**∗7**	**KEY_SEND**, sent by shifted end key
key_seol	**kEOL**	**∗8**	**KEY_SEOL**, sent by shifted clear-line key
key_sexit	**kEXT**	**∗9**	**KEY_SEXIT**, sent by shifted exit key
key_sf	**kind**	**kF**	**KEY_SF**, sent by scroll-forward / down key
key_sfind	**kFND**	**∗0**	**KEY_SFIND**, sent by shifted find key
key_shelp	**kHLP**	**#1**	**KEY_SHELP**, sent by shifted help key
key_shome	**kHOM**	**#2**	**KEY_SHOME**, sent by shifted home key
key_sic	**kIC**	**#3**	**KEY_SIC**, sent by shifted input key
key_sleft	**kLFT**	**#4**	**KEY_SLEFT**, sent by shifted left-arrow key
key_smessage	**kMSG**	**%a**	**KEY_SMESSAGE**, sent by shifted message key
key_smove	**kMOV**	**%b**	**KEY_SMOVE**, sent by shifted move key
key_snext	**kNXT**	**%c**	**KEY_SNEXT**, sent by shifted next key
key_soptions	**kOPT**	**%d**	**KEY_SOPTIONS**, sent by shifted options key
key_sprevious	**kPRV**	**%e**	**KEY_SPREVIOUS**, sent by shifted prev key
key_sprint	**kPRT**	**%f**	**KEY_SPRINT**, sent by shifted print key
key_sr	**kri**	**kR**	**KEY_SR**, sent by scroll-backward / up key
key_sredo	**kRDO**	**%g**	**KEY_SREDO**, sent by shifted redo key
key_sreplace	**kRPL**	**%h**	**KEY_SREPLACE**, sent by shifted replace key
key_sright	**kRIT**	**%i**	**KEY_SRIGHT**, sent by shifted right-arrow key
key_srsume	**kRES**	**%j**	**KEY_SRSUME**, sent by shifted resume key
key_ssave	**kSAV**	**!1**	**KEY_SSAVE**, sent by shifted save key
key_ssuspend	**kSPD**	**!2**	**KEY_SSUSPEND**, sent by shifted suspend key
key_stab	**khts**	**kT**	**KEY_STAB**, sent by set-tab key

key_sundo	**kUND**	**!3**	**KEY_SUNDO**, sent by shifted undo key
key_suspend	**kspd**	**&7**	**KEY_SUSPEND**, sent by suspend key
key_undo	**kund**	**&8**	**KEY_UNDO**, sent by undo key
key_up	**kcuu1**	**ku**	**KEY_UP**, sent by terminal up-arrow key
keypad_local	**rmkx**	**ke**	Out of ‘‘keypad-transmit’’ mode
keypad_xmit	**smkx**	**ks**	Put terminal in ‘‘keypad-transmit’’ mode
lab_f0	**lf0**	**l0**	Labels on function key f0 if not f0
lab_f1	**lf1**	**l1**	Labels on function key f1 if not f1
lab_f2	**lf2**	**l2**	Labels on function key f2 if not f2
lab_f3	**lf3**	**l3**	Labels on function key f3 if not f3
lab_fB	**lfB**	**l4**	Labels on function key fB if not fB
lab_f5	**lf5**	**l5**	Labels on function key f5 if not f5
lab_f6	**lf6**	**l6**	Labels on function key f6 if not f6
lab_f7	**lf7**	**l7**	Labels on function key f7 if not f7
lab_f8	**lf8**	**l8**	Labels on function key f8 if not f8
lab_f9	**lf9**	**l9**	Labels on function key f9 if not f9
lab_f10	**lf10**	**la**	Labels on function key f10 if not f10
label_off	**rmln**	**LF**	Turn off soft labels
label_on	**smln**	**LO**	Turn on soft labels
meta_off	**rmm**	**mo**	Turn off "meta mode"
meta_on	**smm**	**mm**	Turn on "meta mode" (8th bit)
micro_column_address	**mhpa**	**ZY**	Like **column_address** for micro adjustment
micro_down	**mcud1**	**ZZ**	Like **cursor_down** for micro adjustment
micro_left	**mcub1**	**Za**	Like **cursor_left** for micro adjustment
micro_right	**mcuf1**	**Zb**	Like **cursor_right** for micro adjustment
micro_row_address	**mvpa**	**Zc**	Like **row_address** for micro adjustment
micro_up	**mcuu1**	**Zd**	Like **cursor_up** for micro adjustment
newline	**nel**	**nw**	Newline (behaves like **cr** followed by **lf**)
order_of_pins	**porder**	**Ze**	Matches software bits to print-head pins
orig_colors	**oc**	**oc**	Set all color(-pair)s to the original ones
orig_pair	**op**	**op**	Set default color-pair to the original one
pad_char	**pad**	**pc**	Pad character (rather than null)
parm_dch	**dch**	**DC**	Delete #1 chars
parm_delete_line	**dl**	**DL**	Delete #1 lines
parm_down_cursor	**cud**	**DO**	Move down #1 lines.
parm_down_micro	**mcud**	**Zf**	Like **parm_down_cursor** for micro adjust.
parm_ich	**ich**	**IC**	Insert #1 blank chars
parm_index	**indn**	**SF**	Scroll forward #1 lines.
parm_insert_line	**il**	**AL**	Add #1 new blank lines
parm_left_cursor	**cub**	**LE**	Move cursor left #1 spaces

parm_left_micro	**mcub**	**Zg**	Like **parm_left_cursor** for micro adjust.
parm_right_cursor	**cuf**	**RI**	Move right #1 spaces.
parm_right_micro	**mcuf**	**Zh**	Like **parm_right_cursor** for micro adjust.
parm_rindex	**rin**	**SR**	Scroll backward #1 lines.
parm_up_cursor	**cuu**	**UP**	Move cursor up #1 lines.
parm_up_micro	**mcuu**	**Zi**	Like **parm_up_cursor** for micro adjust.
pc_term_options	**pctrm**	**S6**	PC terminal options
pkey_key	**pfkey**	**pk**	Prog funct key #1 to type string #2
pkey_local	**pfloc**	**pl**	Prog funct key #1 to execute string #2
pkey_plab	**pfxl**	**xl**	Prog key #1 to xmit string #2 and show string #3
pkey_xmit	**pfx**	**px**	Prog funct key #1 to xmit string #2
plab_norm	**pln**	**pn**	Prog label #1 to show string #2
print_screen	**mc0**	**ps**	Print contents of the screen
prtr_non	**mc5p**	**pO**	Turn on the printer for #1 bytes
prtr_off	**mc4**	**pf**	Turn off the printer
prtr_on	**mc5**	**po**	Turn on the printer
repeat_char	**rep**	**rp**	Repeat char #1 #2 times
req_for_input	**rfi**	**RF**	Send next input char (for ptys)
reset_1string	**rs1**	**r1**	Reset terminal completely to sane modes
reset_2string	**rs2**	**r2**	Reset terminal completely to sane modes
reset_3string	**rs3**	**r3**	Reset terminal completely to sane modes
reset_file	**rf**	**rf**	Name of file containing reset string
restore_cursor	**rc**	**rc**	Restore cursor to position of last sc
row_address	**vpa**	**cv**	Vertical position absolute
save_cursor	**sc**	**sc**	Save cursor position
scancode_escape	**scesc**	**S7**	Escape for scancode emulation
scroll_forward	**ind**	**sf**	Scroll text up
scroll_reverse	**ri**	**sr**	Scroll text down
select_char_set	**scs**	**Zj**	Select character set
set0_des_seq	**s0ds**	**s0**	Shift into codeset 0 (EUC set 0, ASCII)
set1_des_seq	**s1ds**	**s1**	Shift into codeset 1
set2_des_seq	**s2ds**	**s2**	Shift into codeset 2
set3_des_seq	**s3ds**	**s3**	Shift into codeset 3
set_a_background	**setab**	**AB**	Set background color using ANSI escape
set_a_foreground	**setaf**	**AF**	Set foreground color using ANSI escape
set_attributes	**sgr**	**sa**	Define the video attributes #1-#9
set_background	**setb**	**Sb**	Set current background color
set_bottom_margin	**smgb**	**Zk**	Set bottom margin at current line
set_bottom_margin_parm	**smgbp**	**Zl**	Set bottom margin at line #1 or #2 lines from bottom
set_color_band	**setcolor**	**Yz**	Change to ribbon color #1
set_color_pair	**scp**	**sp**	Set current color-pair

set_foreground	**setf**	**Sf**	Set current foreground color1
set_left_margin	**smgl**	**ML**	Set left margin at current line
set_left_margin_parm	**smglp**	**Zm**	Set left (right) margin at column #1 (#2)
set_lr_margin	**smglr**	**ML**	Sets both left and right margins
set_page_length	**slines**	**YZ**	Set page length to #1 lines (use tparm)
set_right_margin	**smgr**	**MR**	Set right margin at current column
set_right_margin_parm	**smgrp**	**Zn**	Set right margin at column #1
set_tab	**hts**	**st**	Set a tab in all rows, current column
set_tb_margin	**smgtb**	**MT**	Sets both top and bottom margins
set_top_margin	**smgt**	**Zo**	Set top margin at current line
set_top_margin_parm	**smgtp**	**Zp**	Set top (bottom) margin at line #1 (#2)
set_window	**wind**	**wi**	Current window is lines #1-#2 cols #3-#4
start_bit_image	**sbim**	**Zq**	Start printing bit image graphics
start_char_set_def	**scsd**	**Zr**	Start definition of a character set
stop_bit_image	**rbim**	**Zs**	End printing bit image graphics
stop_char_set_def	**rcsd**	**Zt**	End definition of a character set
subscript_characters	**subcs**	**Zu**	List of ‘‘subscript-able’’ characters
superscript_characters	**supcs**	**Zv**	List of ‘‘superscript-able’’ characters
tab	**ht**	**ta**	Tab to next 8-space hardware tab stop
these_cause_cr	**docr**	**Zw**	Printing any of these chars causes **cr**
to_status_line	**tsl**	**ts**	Go to status line, col #1
underline_char	**uc**	**uc**	Underscore one char and move past it
up_half_line	**hu**	**hu**	Half-line up (reverse 1/2 linefeed)
xoff_character	**xoffc**	**XF**	X-off character
xon_character	**xonc**	**XN**	X-on character
zero_motion	**zerom**	**Zx**	No motion for the subsequent character

Sample Entry

The following entry, which describes the AT&T 610 terminal, is among the more complex entries in the **terminfo** file as of this writing.

```
610 | 610bct | ATT610 | att610 | AT&T 610; 80 column; 98key keyboard
    am, eslok, hs, mir, msgr, xenl, xon,
    cols#80, it#8, lh#2, lines#24, lw#8, nlab#8, wsl#80,
    acsc=‘‘aaffggjjkkllmmnnooppqqrrssttuuvvwwxxyyzz{{||}}~~,
    bel=^G, blink=\E[5m, bold=\E[1m, cbt=\E[Z,
    civis=\E[?25l, clear=\E[H\E[J, cnorm=\E[?25h\E[?12l,
    cr=\r, csr=\E[%i%p1%d;%p2%dr, cub=\E[%p1%dD, cub1=\b,
    cud=\E[%p1%dB, cud1=\E[B, cuf=\E[%p1%dC, cuf1=\E[C,
    cup=\E[%i%p1%d;%p2%dH, cuu=\E[%p1%dA, cuu1=\E[A,
    cvvis=\E[?12;25h, dch=\E[%p1%dP, dch1=\E[P, dim=\E[2m,
    dl=\E[%p1%dM, dl1=\E[M, ed=\E[J, el=\E[K, el1=\E[1K,
    flash=\E[?5h$<200>\E[?5l, fsl=\E8, home=\E[H, ht=\t,
    ich=\E[%p1%d@, il=\E[%p1%dL, il1=\E[L, ind=\ED, .ind=\ED$<9>,
    invis=\E[8m,
    is1=\E[8;0 | \E[?3;4;5;13;15l\E[13;20l\E[?7h\E[12h\E(B\E)0,
```

```
    is2=\E[0mˆO, is3=\E(B\E)0, kLFT=\E[\s@, kRIT=\E[\sA,
    kbs=ˆH, kcbt=\E[Z, kclr=\E[2J, kcub1=\E[D, kcud1=\E[B,
    kcuf1=\E[C, kcuu1=\E[A, kf1=\EOc, kf10=\ENp,
    kf11=\ENq, kf12=\ENr, kf13=\ENs, kf14=\ENt, kf2=\EOd,
    kf3=\EOe, kfB=\EOf, kf5=\EOg, kf6=\EOh, kf7=\EOi,
    kf8=\EOj, kf9=\ENo, khome=\E[H, kind=\E[S, kri=\E[T,
    ll=\E[24H, mc4=\E[?4i, mc5=\E[?5i, nel=\EE,
    pfxl=\E[%p1%d;%p2%l%02dq%?%p1%{9}%<%t\s\s\sF%p1%1d\s\s\s\s\s
\s\s\s\s\s\s%;%p2%s,
    pln=\E[%p1%d;0;0;0q%p2%:-16.16s, rc=\E8, rev=\E[7m,
    ri=\EM, rmacs=ˆO, rmir=\E[4l, rmln=\E[2p, rmso=\E[m,
    rmul=\E[m, rs2=\Ec\E[?3l, sc=\E7,
    sgr=\E[0%?%p6%t;1%;%?%p5%t;2%;%?%p2%t;4%;%?%p4%t;5%;
%?%p3%p1% | %t;7%;%?%p7%t;8%;m%?%p9%tˆN%eˆO%;,
    sgr0=\E[mˆO, smacs=ˆN, smir=\E[4h, smln=\E[p,
    smso=\E[7m, smul=\E[4m, tsl=\E7\E[25;%i%p1%dx,
```

Types of Capabilities in the Sample Entry

The sample entry shows the formats for the three types of **terminfo** capabilities listed: Boolean, numeric, and string. All capabilities specified in the **terminfo** source file must be followed by commas, including the last capability in the source file. In **terminfo** source files, capabilities are referenced by their capability names (as shown in the previous tables).

Boolean capabilities are specified simply by their comma separated cap names.

Numeric capabilities are followed by the character '#' and then a positive integer value. Thus, in the sample, **cols** (which shows the number of columns available on a device) is assigned the value **80** for the AT&T 610. (Values for numeric capabilities may be specified in decimal, octal, or hexadecimal, using normal C programming language conventions.)

Finally, string-valued capabilities such as **el** (clear to end of line sequence) are listed by a two- to five-character capname, an '=', and a string ended by the next occurrence of a comma. A delay in milliseconds may appear anywhere in such a capability, preceded by **$** and enclosed in angle brackets, as in **el=\EK$<3>**. Padding characters are supplied by **tput**. The delay can be any of the following: a number, a number followed by an asterisk, such as **5***, a number followed by a slash, such as **5/**, or a number followed by both, such as **5*/**. A '*' shows that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. (In the case of insert characters, the factor is still the number of lines affected. This is always 1 unless the device has **in** and the software uses it.) When a '*' is specified, it is sometimes useful to give a delay of the form **3.5** to specify a delay per unit to tenths of milliseconds. (Only one decimal place is allowed.)

A '/' indicates that the padding is mandatory. If a device has **xon** defined, the padding information is advisory and will only be used for cost estimates or when the device is in raw mode. Mandatory padding will be transmitted regardless of the setting of **xon**. If padding (whether advisory or mandatory) is specified for **bel** or **flash**, however, it will always be used, regardless of whether **xon** is specified.

terminfo offers notation for encoding special characters. Both **\E** and **\e** map to an ESCAPE character, *^x* maps to a control *x* for any appropriate *x*, and the sequences **\n, \l, \r, \t, \b, \f**, and **\s** give a newline, linefeed, return, tab, backspace, formfeed, and space, respectively. Other escapes include: **\^** for caret (^); **** for backslash (\); **\,** for comma (,); **\:** for colon (:); and **\0** for null. (**\0** will actually produce **\200**, which does not terminate a string but behaves as a null character on most devices, providing CS7 is specified. (See **stty**(1)). Finally, characters may be given as three octal digits after a backslash (for example, \123).

Sometimes individual capabilities must be commented out. To do this, put a period before the capability name. For example, see the second **ind** in the example above. Note that capabilities are defined in a left-to-right order and, therefore, a prior definition will override a later definition.

Preparing Descriptions

The most effective way to prepare a device description is by imitating the description of a similar device in **terminfo** and building up a description gradually, using partial descriptions with **vi** to check that they are correct. Be aware that a very unusual device may expose deficiencies in the ability of the **terminfo** file to describe it or the inability of **vi** to work with that device. To test a new device description, set the environment variable **TERMINFO** to the pathname of a directory containing the compiled description you are working on and programs will look there rather than in **/usr/share/lib/terminfo**. To get the padding for insert-line correct (if the device manufacturer did not document it) a severe test is to comment out **xon**, edit a large file at 9600 baud with **vi**, delete 16 or so lines from the middle of the screen, and then press the **u** key several times quickly. If the display is corrupted, more padding is usually needed. A similar test can be used for insert-character.

Section 1-1: Basic Capabilities

The number of columns on each line for the device is given by the **cols** numeric capability. If the device has a screen, then the number of lines on the screen is given by the **lines** capability. If the device wraps around to the beginning of the next line when it reaches the right margin, then it should have the **am** capability. If the terminal can clear its screen, leaving the cursor in the home position, then this is given by the **clear** string capability. If the terminal overstrikes (rather than clearing a position when a character is struck over) then it should have the **os** capability. If the device is a printing terminal, with no soft copy unit, specify both **hc** and **os**. If there is a way to move the cursor to the left edge of the current row, specify this as **cr**.

(Normally this will be carriage return, control M.) If there is a way to produce an audible signal (such as a bell or a beep), specify it as **bel**. If, like most devices, the device uses the xon-xoff flow-control protocol, specify **xon**.

If there is a way to move the cursor one position to the left (such as backspace), that capability should be given as **cub1**. Similarly, sequences to move to the right, up, and down should be given as **cuf1**, **cuu1**, and **cud1**, respectively. These local cursor motions must not alter the text they pass over; for example, you would not normally use ''**cuf1**=\s'' because the space would erase the character moved over.

A very important point here is that the local cursor motions encoded in **terminfo** are undefined at the left and top edges of a screen terminal. Programs should never attempt to backspace around the left edge, unless **bw** is specified, and should never attempt to go up locally off the top. To scroll text up, a program goes to the bottom left corner of the screen and sends the **ind** (index) string.

To scroll text down, a program goes to the top left corner of the screen and sends the **ri** (reverse index) string. The strings **ind** and **ri** are undefined when not on their respective corners of the screen.

Parameterized versions of the scrolling sequences are **indn** and **rin**. These versions have the same semantics as **ind** and **ri**, except that they take one parameter and scroll the number of lines specified by that parameter. They are also undefined except at the appropriate edge of the screen.

The **am** capability tells whether the cursor sticks at the right edge of the screen when text is output, but this does not necessarily apply to a **cuf1** from the last column. Backward motion from the left edge of the screen is possible only when **bw** is specified. In this case, **cub1** will move to the right edge of the previous row. If **bw** is not given, the effect is undefined. This is useful for drawing a box around the edge of the screen, for example. If the device has switch selectable automatic margins, **am** should be specified in the **terminfo** source file. In this case, initialization strings should turn on this option, if possible. If the device has a command that moves to the first column of the next line, that command can be given as **nel** (newline). It does not matter if the command clears the remainder of the current line, so if the device has no **cr** and **lf** it may still be possible to craft a working **nel** out of one or both of them.

These capabilities suffice to describe hardcopy and screen terminals. Thus the AT&T 5320 hardcopy terminal is described as follows:

```
5320|att5320|AT&T 5320 hardcopy terminal,
    am, hc, os,
    cols#132,
    bel=^G, cr=\r, cub1=\b, cnd1=\n,
    dch1=\E[P, dl1=\E[M,
    ind=\n,
```

while the Lear Siegler ADM–3 is described as

adm3 | lsi adm3,
am, bel=^G, clear=^Z, cols#80, cr=^M, cub1=^H,
cud1=^J, ind=^J, lines#24,

Section 1-2: Parameterized Strings

Cursor addressing and other strings requiring parameters are described by a parameterized string capability, with **printf**-like escapes (**%***x*) in it. For example, to address the cursor, the **cup** capability is given, using two parameters: the row and column to address to. (Rows and columns are numbered from zero and refer to the physical screen visible to the user, not to any unseen memory.) If the terminal has memory relative cursor addressing, that can be indicated by **mrcup**.

The parameter mechanism uses a stack and special **%** codes to manipulate the stack in the manner of Reverse Polish Notation (postfix). Typically a sequence will push one of the parameters onto the stack and then print it in some format. Often more complex operations are necessary. Operations are in postfix form with the operands in the usual order. That is, to subtract 5 from the first parameter, one would use **%p1%{5}%–**.

The **%** encodings have the following meanings:

%% outputs '%'

%[[:]*flags*][*width*[.*precision*]][**doxXs**]
as in **printf**, flags are **[–+#]** and space

%c print pop gives %c

%p[1-9]
push *i*th parm

%P[a-z]
set dynamic variable [a-z] to pop

%g[a-z]
get dynamic variable [a-z] and push it

%P[A-Z]
set static variable [a-z] to pop

%g[A-Z]
get static variable [a-z] and push it

%'*c***'** push char constant *c*

%{*nn***}** push decimal constant *nn*

%l push strlen(pop)

%+ %– %* %/ %m
arithmetic (**%m** is mod): push(pop $integer sub 2$ op pop $integer sub 1$)

%& %| %^
bit operations: push(pop $integer sub 2$ op pop $integer sub 1$)

%= %> %<
logical operations: push(pop $integer sub 2$ op pop $integer sub 1$)

%A %O
logical operations: and, or

%! %~ unary operations: push(op pop)

%i (for ANSI terminals) add 1 to first parm, if one parm present, or first two parms, if more than one parm present

%? *expr* **%t** *thenpart* **%e** *elsepart* **%;**
if-then-else, **%e** *elsepart* is optional; else-if's are possible ala Algol 68: **%? c$_1$ %t b$_1$ %e c$_2$ %t b$_2$ %e c$_3$ %t b$_3$ %e c$_4$ %t b$_4$ %e b$_5$%;**
c$_i$ are conditions, b$_i$ are bodies.

If the ''–'' flag is used with ''**%**[**doxXs**]'', then a colon (**:**) must be placed between the ''**%**'' and the ''–'' to differentiate the flag from the binary ''**%**–'' operator, for example ''**%:–16.16s**''.

Consider the Hewlett-Packard 2645, which, to get to row 3 and column 12, needs to be sent **\E&a12c03Y** padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are zero-padded as two digits. Thus its **cup** capability is:

cup=\E&a%p2%2.2dc%p1%2.2dY$<6>

The Micro-Term ACT-IV needs the current row and column sent preceded by a **^T**, with the row and column simply encoded in binary, ''**cup=^T%p1%c%p2%c**''. Devices that use ''**%c**'' need to be able to backspace the cursor (**cub1**), and to move the cursor up one line on the screen (**cuu1**). This is necessary because it is not always safe to transmit **\n**, **^D**, and **\r**, as the system may change or discard them. (The library routines dealing with **terminfo** set tty modes so that tabs are never expanded, so **\t** is safe to send. This turns out to be essential for the Ann Arbor 4080.)

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus ''**cup=\E=%p1%'\s'%+%c%p2%'\s'%+%c**''. After sending ''**\E=**'', this pushes the first parameter, pushes the ASCII value for a space (32), adds them (pushing the sum on the stack in place of the two previous values), and outputs that value as a character. Then the same is done for the second parameter. More complex arithmetic is possible using the stack.

Section 1-3: Cursor Motions

If the terminal has a fast way to home the cursor (to very upper left corner of screen) then this can be given as **home**; similarly a fast way of getting to the lower left-hand corner can be given as **ll**; this may involve going up with **cuu1** from the home position, but a program should never do this itself (unless **ll** does) because it can make no assumption about the effect of moving up from the home position. Note that the home position is the same as addressing to (0,0): to the top left corner of the screen, not of memory.

(Thus, the **\EH** sequence on Hewlett-Packard terminals cannot be used for **home** without losing some of the other features on the terminal.)

If the device has row or column absolute-cursor addressing, these can be given as single parameter capabilities **hpa** (horizontal position absolute) and **vpa** (vertical position absolute). Sometimes these are shorter than the more general two-parameter sequence (as with the Hewlett-Packard 2645) and can be used in preference to **cup**. If there are parameterized local motions (for example, move *n* spaces to the right) these can be given as **cud**, **cub**, **cuf**, and **cuu** with a single parameter indicating how many spaces to move. These are primarily useful if the device does not have **cup**, such as the Tektronix 4025.

If the device needs to be in a special mode when running a program that uses these capabilities, the codes to enter and exit this mode can be given as **smcup** and **rmcup**. This arises, for example, from terminals, such as the Concept, with more than one page of memory. If the device has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the device for cursor addressing to work properly. This is also used for the Tektronix 4025, where **smcup** sets the command character to be the one used by **terminfo**. If the **smcup** sequence will not restore the screen after an **rmcup** sequence is output (to the state prior to outputting **rmcup**), specify **nrrmc**.

Section 1-4: Area Clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as **el**. If the terminal can clear from the beginning of the line to the current position inclusive, leaving the cursor where it is, this should be given as **el1**. If the terminal can clear from the current position to the end of the display, then this should be given as **ed**. **ed** is only defined from the first column of a line. (Thus, it can be simulated by a request to delete a large number of lines, if a true **ed** is not available.)

Section 1-5: Insert/Delete Line

If the terminal can open a new blank line before the line where the cursor is, this should be given as **il1**; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line which the cursor is on, then this should be given as **dl1**; this is done only from the first position on the line to be deleted. Versions of **il1** and **dl1** which take a single parameter and insert or delete that many lines can be given as **il** and **dl**.

If the terminal has a settable destructive scrolling region (like the VT100) the command to set this can be described with the **csr** capability, which takes two parameters: the top and bottom lines of the scrolling region. The cursor position is, alas, undefined after using this command. It is possible to get the effect of insert or delete line using this command — the **sc** and **rc** (save and restore cursor) commands are also useful. Inserting lines at the top or bottom of the screen can also be done using **ri** or **ind** on many terminals without a true insert / delete line, and is often faster even on terminals with those features.

To determine whether a terminal has destructive scrolling regions or non-destructive scrolling regions, create a scrolling region in the middle of the screen, place data on the bottom line of the scrolling region, move the cursor to the top line of the scrolling region, and do a reverse index (**ri**) followed by a delete line (**dl1**) or index (**ind**). If the data that was originally on the bottom line of the scrolling region was restored into the scrolling region by the **dl1** or **ind**, then the terminal has non-destructive scrolling regions. Otherwise, it has destructive scrolling regions. Do not specify **csr** if the terminal has non-destructive scrolling regions, unless **ind**, **ri**, **indn**, **rin**, **dl**, and **dl1** all simulate destructive scrolling.

If the terminal has the ability to define a window as part of memory, which all commands affect, it should be given as the parameterized string **wind**. The four parameters are the starting and ending lines in memory and the starting and ending columns in memory, in that order.

If the terminal can retain display memory above, then the **da** capability should be given; if display memory can be retained below, then **db** should be given. These indicate that deleting a line or scrolling a full screen may bring non-blank lines up from below or that scrolling back with **ri** may bring down non-blank lines.

Section 1-6: Insert/Delete Character

There are two basic kinds of intelligent terminals with respect to insert / delete character operations which can be described using **terminfo.** The most common insert / delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks. You can determine the kind of terminal you have by clearing the screen and then typing text separated by cursor motions. Type ‘‘**abc def**’’ using local cursor motions (not spaces) between the **abc** and the **def**. Then position the cursor before the **abc** and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, then your terminal does not distinguish between blanks and untyped positions. If the **abc** shifts over to the **def** which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability **in**, which stands for ‘‘insert null.’’ While these are two logically separate attributes (one line versus multiline insert mode, and special treatment of untyped spaces) we have seen no terminals whose insert mode cannot be described with the single attribute.

terminfo can describe both terminals that have an insert mode and terminals which send a simple sequence to open a blank position on the current line. Give as **smir** the sequence to get into insert mode. Give as **rmir** the sequence to leave insert mode. Now give as **ich1** any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give **ich1**; terminals that send a sequence to open a screen position should give it here. (If your terminal has both, insert mode is

usually preferable to **ich1**. Do not give both unless the terminal actually requires both to be used in combination.) If post-insert padding is needed, give this as a number of milliseconds padding in **ip** (a string option). Any other sequence which may need to be sent after an insert of a single character may also be given in **ip**. If your terminal needs both to be placed into an 'insert mode' and a special code to precede each inserted character, then both **smir**/rmir and **ich1** can be given, and both will be used. The **ich** capability, with one parameter, *n*, will insert *n* blanks.

If padding is necessary between characters typed while not in insert mode, give this as a number of milliseconds padding in **rmp**.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (for example, if there is a tab after the insertion position). If your terminal allows motion while in insert mode you can give the capability **mir** to speed up inserting in this case. Omitting **mir** will affect only speed. Some terminals (notably Datamedia's) must not have **mir** because of the way their insert mode works.

Finally, you can specify **dch1** to delete a single character, **dch** with one parameter, *n*, to delete *n* characters, and delete mode by giving **smdc** and **rmdc** to enter and exit delete mode (any mode the terminal needs to be placed in for **dch1** to work).

A command to erase *n* characters (equivalent to outputting *n* blanks without moving the cursor) can be given as **ech** with one parameter.

Section 1-7: Highlighting, Underlining, and Visible Bells

Your device may have one or more kinds of display attributes that allow you to highlight selected characters when they appear on the screen. The following display modes (shown with the names by which they are set) may be available: a blinking screen (**blink**), bold or extra-bright characters (**bold**), dim or half-bright characters (**dim**), blanking or invisible text (**invis**), protected text (**prot**), a reverse-video screen (**rev**), and an alternate character set (**smacs** to enter this mode and **rmacs** to exit it). (If a command is necessary before you can enter alternate character set mode, give the sequence in **enacs** or "enable alternate-character-set" mode.) Turning on any of these modes singly may or may not turn off other modes.

sgr0 should be used to turn off all video enhancement capabilities. It should always be specified because it represents the only way to turn off some capabilities, such as **dim** or **blink**.

You should choose one display method as *standout mode* and use it to highlight error messages and other kinds of text to which you want to draw attention. Choose a form of display that provides strong contrast but that is easy on the eyes. (We recommend reverse-video plus half-bright or reverse-video alone.) The sequences to enter and exit standout mode are given as **smso** and **rmso**, respectively. If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, as the TVI 912 and Teleray 1061 do, then **xmc** should be given to tell how many spaces are left.

Sequences to begin underlining and end underlining can be specified as **smul** and **rmul ,** respectively. If the device has a sequence to underline the current character and to move the cursor one space to the right (such as the Micro-Term MIME), this sequence can be specified as **uc**.

Terminals with the ‘‘magic cookie’’ glitch (**xmc)** deposit special ‘‘cookies’’ when they receive mode-setting sequences, which affect the display algorithm rather than having extra bits for each character. Some terminals, such as the Hewlett-Packard 2621, automatically leave standout mode when they move to a new line or the cursor is addressed. Programs using standout mode should exit standout mode before moving the cursor or sending a newline, unless the **msgr** capability, asserting that it is safe to move in standout mode, is present.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement), then this can be given as **flash**; it must not move the cursor. A good flash can be done by changing the screen into reverse video, pad for 200 ms, then return the screen to normal video.

If the cursor needs to be made more visible than normal when it is not on the bottom line (to make, for example, a non-blinking underline into an easier to find block or blinking underline) give this sequence as **cvvis**. The boolean **chts** should also be given. If there is a way to make the cursor completely invisible, give that as **civis**. The capability **cnorm** should be given which undoes the effects of either of these modes.

If your terminal generates underlined characters by using the underline character (with no special sequences needed) even though it does not otherwise overstrike characters, then you should specify the capability **ul**. For devices on which a character overstriking another leaves both characters on the screen, specify the capability **os**. If overstrikes are erasable with a blank, then this should be indicated by specifying **eo**.

If there is a sequence to set arbitrary combinations of modes, this should be given as **sgr** (set attributes), taking nine parameters. Each parameter is either **0** or non-zero, as the corresponding attribute is on or off. The nine parameters are, in order: standout, underline, reverse, blink, dim, bold, blank, protect, alternate character set. Not all modes need to be supported by **sgr**; only those for which corresponding separate attribute commands exist should be supported. For example, let’s assume that the terminal in question needs the following escape sequences to turn on various modes.

tparm parameter	attribute	escape sequence
	none	**\E[0m**
p1	standout	**\E[0;4;7m**
p2	underline	**\E[0;3m**
p3	reverse	**\E[0;4m**
p4	blink	**\E[0;5m**
p5	dim	**\E[0;7m**
p6	bold	**\E[0;3;4m**
p7	invis	**\E[0;8m**
p8	protect	not available
p9	altcharset	**^O** (off) **^N** (on)

Note that each escape sequence requires a **0** to turn off other modes before turning on its own mode. Also note that, as suggested above, *standout* is set up to be the combination of *reverse* and *dim*. Also, because this terminal has no *bold* mode, *bold* is set up as the combination of *reverse* and *underline*. In addition, to allow combinations, such as *underline+blink*, the sequence to use would be **\E[0;3;5m**. The terminal doesn't have *protect* mode, either, but that cannot be simulated in any way, so **p8** is ignored. The *altcharset* mode is different in that it is either **^O** or **^N**, depending on whether it is off or on. If all modes were to be turned on, the sequence would be **\E[0;3;4;5;7;8m^N**.

Now look at when different sequences are output. For example, **;3** is output when either **p2** or **p6** is true, that is, if either *underline* or *bold* modes are turned on.
Writing out the above sequences, along with their dependencies, gives the following:

sequence	when to output	terminfo translation
\E[0	always	**\E[0**
;3	if **p2** or **p6**	**%?%p2%p6%|%t;3%;**
;4	if **p1** or **p3** or **p6**	**%?%p1%p3%|%p6%|%t;4%;**
;5	if **p4**	**%?%p4%t;5%;**
;7	if **p1** or **p5**	**%?%p1%p5%|%t;7%;**
;8	if **p7**	**%?%p7%t;8%;**
m	always	**m**
^N or **^O**	if **p9** **^N**, else **^O**	**%?%p9%t^N%e^O%;**

Putting this all together into the **sgr** sequence gives:

sgr=\E[0%?%p2%p6%|%t;3%;%?%p1%p3%|%p6%
|%t;4%;%?%p5%t;5%;%?%p1%p5%
|%t;7%;%?%p7%t;8%;m%?%p9%t^N%e^O%;,

Remember that **sgr** and **sgr0** must always be specified.

Section 1-8: Keypad

If the device has a keypad that transmits sequences when the keys are pressed, this information can also be specified. Note that it is not possible to handle devices where the keypad only works in local (this applies, for example, to the unshifted Hewlett-Packard

2621 keys). If the keypad can be set to transmit or not transmit, specify these sequences as **smkx** and **rmkx**. Otherwise the keypad is assumed to always transmit.

The sequences sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kcub1, kcuf1, kcuu1, kcud1,** and **khome**, respectively. If there are function keys such as f0, f1, ..., f63, the sequences they send can be specified as **kf0, kf1, ..., kf63**. If the first 11 keys have labels other than the default f0 through f10, the labels can be given as **lf0, lf1, ..., lf10**. The codes transmitted by certain other special keys can be given: **kll** (home down), **kbs** (backspace), **ktbc** (clear all tabs), **kctab** (clear the tab stop in this column), **kclr** (clear screen or erase key), **kdch1** (delete character), **kdl1** (delete line), **krmir** (exit insert mode), **kel** (clear to end of line), **ked** (clear to end of screen), **kich1** (insert character or enter insert mode), **kil1** (insert line), **knp** (next page), **kpp** (previous page), **kind** (scroll forward / down), **kri** (scroll backward / up), **khts** (set a tab stop in this column). In addition, if the keypad has a 3 by 3 array of keys including the four arrow keys, the other five keys can be given as **ka1**, **ka3**, **kb2**, **kc1**, and **kc3**. These keys are useful when the effects of a 3 by 3 directional pad are needed. Further keys are defined above in the capabilities list.

Strings to program function keys can be specified as **pfkey**, **pfloc**, and **pfx**. A string to program screen labels should be specified as **pln**. Each of these strings takes two parameters: a function key identifier and a string to program it with. **pfkey** causes pressing the given key to be the same as the user typing the given string; **pfloc** causes the string to be executed by the terminal in local mode; and **pfx** causes the string to be transmitted to the computer. The capabilities **nlab**, **lw** and **lh** define the number of programmable screen labels and their width and height. If there are commands to turn the labels on and off, give them in **smln** and **rmln**. **smln** is normally output after one or more **pln** sequences to make sure that the change becomes visible.

Section 1-9: Tabs and Initialization

If the device has hardware tabs, the command to advance to the next tab stop can be given as **ht** (usually control I). A ''backtab'' command that moves leftward to the next tab stop can be given as **cbt**. By convention, if tty modes show that tabs are being expanded by the computer rather than being sent to the device, programs should not use **ht** or **cbt** (even if they are present) because the user may not have the tab stops properly set. If the device has hardware tabs that are initially set every *n* spaces when the device is powered up, the numeric parameter **it** is given, showing the number of spaces the tabs are set to. This is normally used by **tput init** (see **tput**(1)) to determine whether to set the mode for hardware tab expansion and whether to set the tab stops. If the device has tab stops that can be saved in nonvolatile memory, the **terminfo** description can assume that they are properly set. If there are commands to set and clear tab stops, they can be given as **tbc** (clear all tab stops) and **hts** (set a tab stop in the current column of every row).

Other capabilities include: **is1**, **is2**, and **is3**, initialization strings for the device; **iprog**, the path name of a program to be run to initialize the device; and **if**, the name of a file containing long initialization strings. These strings are expected to set the device into modes

consistent with the rest of the **terminfo** description. They must be sent to the device each time the user logs in and be output in the following order: run the program **iprog**; output **is1**; output **is2**; set the margins using **mgc**, **smgl** and **smgr**; set the tabs using **tbc** and **hts**; print the file **if**; and finally output **is3**. This is usually done using the **init** option of **tput**.

Most initialization is done with **is2**. Special device modes can be set up without duplicating strings by putting the common sequences in **is2** and special cases in **is1** and **is3**. Sequences that do a reset from a totally unknown state can be given as **rs1**, **rs2**, **rf**, and **rs3**, analogous to **is1**, **is2**, **is3**, and **if**. (The method using files, **if** and **rf**, is used for a few terminals, from **/usr/share/lib/tabset/***; however, the recommended method is to use the initialization and reset strings.) These strings are output by **tput** reset, which is used when the terminal gets into a wedged state. Commands are normally placed in **rs1**, **rs2**, **rs3**, and **rf** only if they produce annoying effects on the screen and are not necessary when logging in. For example, the command to set a terminal into 80-column mode would normally be part of **is2**, but on some terminals it causes an annoying glitch on the screen and is not normally needed because the terminal is usually already in 80-column mode.

If a more complex sequence is needed to set the tabs than can be described by using **tbc** and **hts**, the sequence can be placed in **is2** or **if**.

Any margin can be cleared with **mgc**. (For instructions on how to specify commands to set and clear margins, see "Margins" below under "PRINTER CAPABILITIES.")

Section 1-10: Delays

Certain capabilities control padding in the **tty** driver. These are primarily needed by hard-copy terminals, and are used by **tput init** to set tty modes appropriately. Delays embedded in the capabilities **cr**, **ind**, **cub1**, **ff**, and **tab** can be used to set the appropriate delay bits to be set in the tty driver. If **pb** (padding baud rate) is given, these values can be ignored at baud rates below the value of **pb**.

Section 1-11: Status Lines

If the terminal has an extra ‘‘status line’’ that is not normally used by software, this fact can be indicated. If the status line is viewed as an extra line below the bottom line, into which one can cursor address normally (such as the Heathkit h19’s 25th line, or the 24th line of a VT100 which is set to a 23-line scrolling region), the capability **hs** should be given. Special strings that go to a given column of the status line and return from the status line can be given as **tsl** and **fsl**. (**fsl** must leave the cursor position in the same place it was before **tsl**. If necessary, the **sc** and **rc** strings can be included in **tsl** and **fsl** to get this effect.) The capability **tsl** takes one parameter, which is the column number of the status line the cursor is to be moved to.

If escape sequences and other special commands, such as tab, work while in the status line, the flag **eslok** can be given. A string which turns off the status line (or otherwise erases its contents) should be given as **dsl**. If the terminal has commands to save and restore the position of the cursor, give them as **sc** and **rc**. The status line is normally assumed to be the same width as the rest of the screen, for example, **cols**. If the status

line is a different width (possibly because the terminal does not allow an entire line to be loaded) the width, in columns, can be indicated with the numeric parameter **wsl**.

Section 1-12: Line Graphics

If the device has a line drawing alternate character set, the mapping of glyph to character would be given in **acsc**. The definition of this string is based on the alternate character set used in the DEC VT100 terminal, extended slightly with some characters from the AT&T 4410v1 terminal.

glyph name	vt100+ character
arrow pointing right	**+**
arrow pointing left	**,**
arrow pointing down	**.**
solid square block	**0**
lantern symbol	**I**
arrow pointing up	**–**
diamond	**‘**
checker board (stipple)	**a**
degree symbol	**f**
plus / minus	**g**
board of squares	**h**
lower right corner	**j**
upper right corner	**k**
upper left corner	**l**
lower left corner	**m**
plus	**n**
scan line 1	**o**
horizontal line	**q**
scan line 9	**s**
left tee (├)	**t**
right tee (┤)	**u**
bottom tee (┴)	**v**
top tee (┬)	**w**
vertical line	**x**
bullet	**~**

The best way to describe a new device's line graphics set is to add a third column to the above table with the characters for the new device that produce the appropriate glyph when the device is in the alternate character set mode. For example,

glyph name	vt100+ char	new tty char
upper left corner	**l**	**R**
lower left corner	**m**	**F**
upper right corner	**k**	**T**
lower right corner	**j**	**G**
horizontal line	**q**	**,**
vertical line	**x**	**.**

Now write down the characters left to right, as in ''**acsc=lRmFkTjGq\,x.**''.

In addition, **terminfo** allows you to define multiple character sets. See Section 2-5 for details.

Section 1-13: Color Manipulation

Let us define two methods of color manipulation: the Tektronix method and the HP method. The Tektronix method uses a set of N predefined colors (usually 8) from which a user can select "current" foreground and background colors. Thus a terminal can support up to N colors mixed into N∗N color-pairs to be displayed on the screen at the same time. When using an HP method the user cannot define the foreground independently of the background, or vice-versa. Instead, the user must define an entire color-pair at once. Up to M color-pairs, made from 2∗M different colors, can be defined this way. Most existing color terminals belong to one of these two classes of terminals.

The numeric variables **colors** and **pairs** define the number of colors and color-pairs that can be displayed on the screen at the same time. If a terminal can change the definition of a color (for example, the Tektronix 4100 and 4200 series terminals), this should be specified with **ccc** (can change color). To change the definition of a color (Tektronix 4200 method), use **initc** (initialize color). It requires four arguments: color number (ranging from 0 to **colors**–1) and three RGB (red, green, and blue) values or three HLS colors (Hue, Lightness, Saturation). Ranges of RGB and HLS values are terminal dependent.

Tektronix 4100 series terminals only use HLS color notation. For such terminals (or dual-mode terminals to be operated in HLS mode) one must define a boolean variable **hls**; that would instruct the **curses init_color** routine to convert its RGB arguments to HLS before sending them to the terminal. The last three arguments to the **initc** string would then be HLS values.

If a terminal can change the definitions of colors, but uses a color notation different from RGB and HLS, a mapping to either RGB or HLS must be developed.

To set current foreground or background to a given color, use **setaf** (set ANSI foreground) and **setab** (set ANSI background). They require one parameter: the number of the color. To initialize a color-pair (HP method), use **initp** (initialize pair). It requires

seven parameters: the number of a color-pair (range=0 to **pairs**–1), and six RGB values: three for the foreground followed by three for the background. (Each of these groups of three should be in the order RGB.) When **initc** or **initp** are used, RGB or HLS arguments should be in the order "red, green, blue" or "hue, lightness, saturation"), respectively. To make a color-pair current, use **scp** (set color-pair). It takes one parameter, the number of a color-pair.

Some terminals (for example, most color terminal emulators for PCs) erase areas of the screen with current background color. In such cases, **bce** (background color erase) should be defined. The variable **op** (original pair) contains a sequence for setting the foreground and the background colors to what they were at the terminal start-up time. Similarly, **oc** (original colors) contains a control sequence for setting all colors (for the Tektronix method) or color-pairs (for the HP method) to the values they had at the terminal start-up time.

Some color terminals substitute color for video attributes. Such video attributes should not be combined with colors. Information about these video attributes should be packed into the **ncv** (no color video) variable. There is a one-to-one correspondence between the nine least significant bits of that variable and the video attributes. The following table depicts this correspondence.

Attribute	Bit Position	Decimal Value
A_STANDOUT	0	1
A_UNDERLINE	1	2
A_REVERSE	2	4
A_BLINK	3	8
A_DIM	4	16
A_BOLD	5	32
A_INVIS	6	64
A_PROTECT	7	128
A_ALTCHARSET	8	256

When a particular video attribute should not be used with colors, the corresponding **ncv** bit should be set to 1; otherwise it should be set to zero. To determine the information to pack into the **ncv** variable, you must add together the decimal values corresponding to those attributes that cannot coexist with colors. For example, if the terminal uses colors to simulate reverse video (bit number 2 and decimal value 4) and bold (bit number 5 and decimal value 32), the resulting value for **ncv** will be 36 (4 + 32).

Section 1-14: Miscellaneous

If the terminal requires other than a null (zero) character as a pad, then this can be given as **pad**. Only the first character of the **pad** string is used. If the terminal does not have a pad character, specify **npc**.

If the terminal can move up or down half a line, this can be indicated with **hu** (half-line up) and **hd** (half-line down). This is primarily useful for superscripts and subscripts on hardcopy terminals. If a hardcopy terminal can eject to the next page (form feed), give this as **ff** (usually control L).

If there is a command to repeat a given character a given number of times (to save time transmitting a large number of identical characters) this can be indicated with the parameterized string **rep**. The first parameter is the character to be repeated and the second is the number of times to repeat it. Thus, **tparm(repeat_char, 'x', 10)** is the same as **xxxxxxxxxx.**

If the terminal has a settable command character, such as the Tektronix 4025, this can be indicated with **cmdch**. A prototype command character is chosen which is used in all capabilities. This character is given in the **cmdch** capability to identify it. The following convention is supported on some systems: If the environment variable **CC** exists, all occurrences of the prototype character are replaced with the character in **CC**.

Terminal descriptions that do not represent a specific kind of known terminal, such as *switch*, *dialup*, *patch*, and *network*, should include the **gn** (generic) capability so that programs can complain that they do not know how to talk to the terminal. (This capability does not apply to *virtual* terminal descriptions for which the escape sequences are known.) If the terminal is one of those supported by the system virtual terminal protocol, the terminal number can be given as **vt**. A line-turn-around sequence to be transmitted before doing reads should be specified in **rfi**.

If the device uses xon/xoff handshaking for flow control, give **xon**. Padding information should still be included so that routines can make better decisions about costs, but actual pad characters will not be transmitted. Sequences to turn on and off xon/xoff handshaking may be given in **smxon** and **rmxon**. If the characters used for handshaking are not **^S** and **^Q**, they may be specified with **xonc** and **xoffc**.

If the terminal has a ''meta key'' which acts as a shift key, setting the 8th bit of any character transmitted, this fact can be indicated with **km**. Otherwise, software will assume that the 8th bit is parity and it will usually be cleared. If strings exist to turn this ''meta mode'' on and off, they can be given as **smm** and **rmm**.

If the terminal has more lines of memory than will fit on the screen at once, the number of lines of memory can be indicated with **lm**. A value of **lm**#0 indicates that the number of lines is not fixed, but that there is still more memory than fits on the screen.

Media copy strings which control an auxiliary printer connected to the terminal can be given as **mc0**: print the contents of the screen, **mc4**: turn off the printer, and **mc5**: turn on the printer. When the printer is on, all text sent to the terminal will be sent to the printer.

A variation, **mc5p**, takes one parameter, and leaves the printer on for as many characters as the value of the parameter, then turns the printer off. The parameter should not exceed 255. If the text is not displayed on the terminal screen when the printer is on, specify **mc5i** (silent printer). All text, including **mc4**, is transparently passed to the printer while an **mc5p** is in effect.

Section 1-15: Special Cases

The working model used by **terminfo** fits most terminals reasonably well. However, some terminals do not completely match that model, requiring special support by **terminfo**. These are not meant to be construed as deficiencies in the terminals; they are just differences between the working model and the actual hardware. They may be unusual devices or, for some reason, do not have all the features of the **terminfo** model implemented.

Terminals that cannot display tilde (~) characters, such as certain Hazeltine terminals, should indicate **hz**.

Terminals that ignore a linefeed immediately after an **am** wrap, such as the Concept 100, should indicate **xenl**. Those terminals whose cursor remains on the right-most column until another character has been received, rather than wrapping immediately upon receiving the right-most character, such as the VT100, should also indicate **xenl**.

If **el** is required to get rid of standout (instead of writing normal text on top of it), **xhp** should be given.

Those Teleray terminals whose tabs turn all characters moved over to blanks, should indicate **xt** (destructive tabs). This capability is also taken to mean that it is not possible to position the cursor on top of a ''magic cookie.'' Therefore, to erase standout mode, it is necessary, instead, to use delete and insert line.

Those Beehive Superbee terminals which do not transmit the escape or control–C characters, should specify **xsb**, indicating that the f1 key is to be used for escape and the f2 key for control C.

Section 1-16: Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **use** can be given with the name of the similar terminal. The capabilities given before **use** override those in the terminal type invoked by **use**. A capability can be canceled by placing *xx***@** to the left of the capability definition, where *xx* is the capability. For example, the entry

att4424-2|Teletype 4424 in display function group ii,
rev@, sgr@, smul@, use=att4424,

defines an AT&T 4424 terminal that does not have the **rev**, **sgr**, and **smul** capabilities, and hence cannot do highlighting. This is useful for different modes for a terminal, or for different user preferences. More than one **use** capability may be given.

PART 2: PRINTER CAPABILITIES

The **terminfo** database allows you to define capabilities of printers as well as terminals. To find out what capabilities are available for printers as well as for terminals, see the two lists under "DEVICE CAPABILITIES" that list capabilities by variable and by capability name.

Section 2-1: Rounding Values

Because parameterized string capabilities work only with integer values, we recommend that **terminfo** designers create strings that expect numeric values that have been rounded. Application designers should note this and should always round values to the nearest integer before using them with a parameterized string capability.

Section 2-2: Printer Resolution

A printer's resolution is defined to be the smallest spacing of characters it can achieve. In general printers have independent resolution horizontally and vertically. Thus the vertical resolution of a printer can be determined by measuring the smallest achievable distance between consecutive printing baselines, while the horizontal resolution can be determined by measuring the smallest achievable distance between the left-most edges of consecutive printed, identical, characters.

All printers are assumed to be capable of printing with a uniform horizontal and vertical resolution. The view of printing that **terminfo** currently presents is one of printing inside a uniform matrix: All characters are printed at fixed positions relative to each ''cell'' in the matrix; furthermore, each cell has the same size given by the smallest horizontal and vertical step sizes dictated by the resolution. (The cell size can be changed as will be seen later.)

Many printers are capable of ''proportional printing,'' where the horizontal spacing depends on the size of the character last printed. **terminfo** does not make use of this capability, although it does provide enough capability definitions to allow an application to simulate proportional printing.

A printer must not only be able to print characters as close together as the horizontal and vertical resolutions suggest, but also of ''moving'' to a position an integral multiple of the smallest distance away from a previous position. Thus printed characters can be spaced apart a distance that is an integral multiple of the smallest distance, up to the length or width of a single page.

Some printers can have different resolutions depending on different ''modes.'' In ''normal mode,'' the existing **terminfo** capabilities are assumed to work on columns and lines, just like a video terminal. Thus the old **lines** capability would give the length of a page in lines, and the **cols** capability would give the width of a page in columns. In ''micro mode,'' many **terminfo** capabilities work on increments of lines and columns. With some printers the micro mode may be concomitant with normal mode, so that all the capabilities work at the same time.

Section 2-3: Specifying Printer Resolution

The printing resolution of a printer is given in several ways. Each specifies the resolution as the number of smallest steps per distance:

Specification of Printer Resolution
Characteristic Number of Smallest Steps

orhi	Steps per inch horizontally
orvi	Steps per inch vertically
orc	Steps per column
orl	Steps per line

When printing in normal mode, each character printed causes movement to the next column, except in special cases described later; the distance moved is the same as the per-column resolution. Some printers cause an automatic movement to the next line when a character is printed in the rightmost position; the distance moved vertically is the same as the per-line resolution.
When printing in micro mode, these distances can be different, and may be zero for some printers.

Specification of Printer Resolution
Automatic Motion after Printing

Normal Mode:	
orc	Steps moved horizontally
orl	Steps moved vertically
Micro Mode:	
mcs	Steps moved horizontally
mls	Steps moved vertically

Some printers are capable of printing wide characters. The distance moved when a wide character is printed in normal mode may be different from when a regular width character is printed. The distance moved when a wide character is printed in micro mode may also be different from when a regular character is printed in micro mode, but the differences are assumed to be related: If the distance moved for a regular character is the same whether in normal mode or micro mode (**mcs**=**orc**), then the distance moved for a wide character is also the same whether in normal mode or micro mode. This doesn't mean the normal character distance is necessarily the same as the wide character distance, just that the distances don't change with a change in normal to micro mode. However, if the distance moved for a regular character is different in micro mode from the distance moved in normal mode (**mcs**<**orc**), the micro mode distance is assumed to be the same for a wide character printed in micro mode, as the table below shows.

Specification of Printer Resolution
Automatic Motion after Printing Wide Character

*Normal Mode or Micro Mode (***mcs** = **orc**):
widcs Steps moved horizontally

*Micro Mode (***mcs** < **orc**):
mcs Steps moved horizontally

There may be control sequences to change the number of columns per inch (the character pitch) and to change the number of lines per inch (the line pitch). If these are used, the resolution of the printer changes, but the type of change depends on the printer:

Specification of Printer Resolution
Changing the Character / Line Pitches

cpi	Change character pitch
cpix	If set, **cpi** changes **orhi**, otherwise changes **orc**
lpi	Change line pitch
lpix	If set, **lpi** changes **orvi**, otherwise changes **orl**
chr	Change steps per column
cvr	Change steps per line

The **cpi** and **lpi** string capabilities are each used with a single argument, the pitch in columns (or characters) and lines per inch, respectively. The **chr** and **cvr** string capabilities are each used with a single argument, the number of steps per column and line, respectively.

Using any of the control sequences in these strings will imply a change in some of the values of **orc**, **orhi**, **orl**, and **orvi**. Also, the distance moved when a wide character is printed, **widcs**, changes in relation to **orc**. The distance moved when a character is printed in micro mode, **mcs**, changes similarly, with one exception: if the distance is 0 or 1, then no change is assumed (see items marked with † in the following table).

Programs that use **cpi**, **lpi**, **chr**, or **cvr** should recalculate the printer resolution (and should recalculate other values— see "Effect of Changing Printing Resolution" under "Dot-Mapped Graphics").

Specification of Printer Resolution
Effects of Changing the Character / Line Pitches

Before	*After*
Using **cpi** with **cpix** clear:	
$bold orhi '$	**orhi**
$bold orc '$	$bold orc = bold orhi over V sub italic cpi$
Using **cpi** with **cpix** set:	
$bold orhi '$	$bold orhi = bold orc cdot V sub italic cpi$
$bold orc '$	$bold orc$
Using **lpi** with **lpix** clear:	
$bold orvi '$	$bold orvi$
$bold orl '$	$bold orl = bold orvi over V sub italic lpi$
Using **lpi** with **lpix** set:	
$bold orvi '$	$bold orvi = bold orl cdot V sub italic lpi$
$bold orl '$	$bold orl$
Using **chr:**	
$bold orhi '$	$bold orhi$
$bold orc '$	$V sub italic chr$
Using **cvr:**	
$bold orvi '$	$bold orvi$
$bold orl '$	$V sub italic cvr$
Using **cpi** or **chr:**	
$bold widcs '$	$bold widcs = bold {widcs '} bold orc over { bold {orc '} }$
$bold mcs '$	$bold mcs = bold {mcs '} bold orc over { bold {orc '} }$

$V sub italic cpi$, $V sub italic lpi$, $V sub italic chr$, and $V sub italic cvr$ are the arguments used with **cpi**, **lpi**, **chr**, and **cvr**, respectively. The prime marks (') indicate the old values.

Section 2-4: Capabilities that Cause Movement

In the following descriptions, ''movement'' refers to the motion of the ''current position.'' With video terminals this would be the cursor; with some printers this is the carriage position. Other printers have different equivalents. In general, the current position is where a character would be displayed if printed.

terminfo has string capabilities for control sequences that cause movement a number of full columns or lines. It also has equivalent string capabilities for control sequences that cause movement a number of smallest steps.

String Capabilities for Motion

mcub1	Move 1 step left
mcuf1	Move 1 step right
mcuu1	Move 1 step up
mcud1	Move 1 step down
mcub	Move *N* steps left
mcuf	Move *N* steps right
mcuu	Move *N* steps up
mcud	Move *N* steps down
mhpa	Move *N* steps from the left
mvpa	Move *N* steps from the top

The latter six strings are each used with a single argument, *N*.

Sometimes the motion is limited to less than the width or length of a page. Also, some printers don't accept absolute motion to the left of the current position. **terminfo** has capabilities for specifying these limits.

Limits to Motion

mjump	Limit on use of **mcub1**, **mcuf1**, **mcuu1**, **mcud1**
maddr	Limit on use of **mhpa**, **mvpa**
xhpa	If set, **hpa** and **mhpa** can't move left
xvpa	If set, **vpa** and **mvpa** can't move up

If a printer needs to be in a ''micro mode'' for the motion capabilities described above to work, there are string capabilities defined to contain the control sequence to enter and exit this mode. A boolean is available for those printers where using a carriage return causes an automatic return to normal mode.

Entering/Exiting Micro Mode

smicm	Enter micro mode
rmicm	Exit micro mode
crxm	Using **cr** exits micro mode

The movement made when a character is printed in the rightmost position varies among printers. Some make no movement, some move to the beginning of the next line, others move to the beginning of the same line. **terminfo** has boolean capabilities for describing all three cases.

What Happens After Character Printed in Rightmost Position

sam	Automatic move to beginning of same line

Some printers can be put in a mode where the normal direction of motion is reversed. This mode can be especially useful when there are no capabilities for leftward or upward motion, because those capabilities can be built from the motion reversal capability and the rightward or downward motion capabilities. It is best to leave it up to an application

to build the leftward or upward capabilities, though, and not enter them in the **terminfo** database. This allows several reverse motions to be strung together without intervening wasted steps that leave and reenter reverse mode.

Entering / Exiting Reverse Modes

slm	Reverse sense of horizontal motions
rlm	Restore sense of horizontal motions
sum	Reverse sense of vertical motions
rum	Restore sense of vertical motions

While sense of horizontal motions reversed:

mcub1	Move 1 step right
mcuf1	Move 1 step left
mcub	Move *N* steps right
mcuf	Move *N* steps left
cub1	Move 1 column right
cuf1	Move 1 column left
cub	Move *N* columns right
cuf	Move *N* columns left

While sense of vertical motions reversed:

mcuu1	Move 1 step down
mcud1	Move 1 step up
mcuu	Move *N* steps down
mcud	Move *N* steps up
cuu1	Move 1 line down
cud1	Move 1 line up
cuu	Move *N* lines down
cud	Move *N* lines up

The reverse motion modes should not affect the **mvpa** and **mhpa** absolute motion capabilities. The reverse vertical motion mode should, however, also reverse the action of the line ‘‘wrapping’’ that occurs when a character is printed in the right-most position. Thus printers that have the standard **terminfo** capability **am** defined should experience motion to the beginning of the previous line when a character is printed in the right-most position under reverse vertical motion mode.

The action when any other motion capabilities are used in reverse motion modes is not defined; thus, programs must exit reverse motion modes before using other motion capabilities.

Two miscellaneous capabilities complete the list of new motion capabilities. One of these is needed for printers that move the current position to the beginning of a line when certain control characters, such as ‘‘line-feed’’ or ‘‘form-feed,’’ are used. The other is used for the capability of suspending the motion that normally occurs after printing a character.

Miscellaneous Motion Strings

docr	List of control characters causing **cr**
zerom	Prevent auto motion after printing next single character

Margins

terminfo provides two strings for setting margins on terminals: one for the left and one for the right margin. Printers, however, have two additional margins, for the top and bottom margins of each page. Furthermore, some printers require not using motion strings to move the current position to a margin and then fixing the margin there, but require the specification of where a margin should be regardless of the current position. Therefore **terminfo** offers six additional strings for defining margins with printers.

Setting Margins

smgl	Set left margin at current column
smgr	Set right margin at current column
smgb	Set bottom margin at current line
smgt	Set top margin at current line
smgbp	Set bottom margin at line *N*
smglp	Set left margin at column *N*
smgrp	Set right margin at column *N*
smgtp	Set top margin at line *N*

The last four strings are used with one or more arguments that give the position of the margin or margins to set. If both of **smglp** and **smgrp** are set, each is used with a single argument, *N*, that gives the column number of the left and right margin, respectively. If both of **smgtp** and **smgbp** are set, each is used to set the top and bottom margin, respectively: **smgtp** is used with a single argument, *N*, the line number of the top margin; however, **smgbp** is used with two arguments, *N* and *M*, that give the line number of the bottom margin, the first counting from the top of the page and the second counting from the bottom. This accommodates the two styles of specifying the bottom margin in different manufacturers' printers. When coding a **terminfo** entry for a printer that has a settable bottom margin, only the first or second parameter should be used, depending on the printer. When writing an application that uses **smgbp** to set the bottom margin, both arguments must be given.

If only one of **smglp** and **smgrp** is set, then it is used with two arguments, the column number of the left and right margins, in that order. Likewise, if only one of **smgtp** and **smgbp** is set, then it is used with two arguments that give the top and bottom margins, in that order, counting from the top of the page. Thus when coding a **terminfo** entry for a printer that requires setting both left and right or top and bottom margins simultaneously, only one of **smglp** and **smgrp** or **smgtp** and **smgbp** should be defined; the other should be left blank. When writing an application that uses these string capabilities, the pairs should be first checked to see if each in the pair is set or only one is set, and should then be used accordingly.

In counting lines or columns, line zero is the top line and column zero is the left-most column. A zero value for the second argument with **smgbp** means the bottom line of the page.

All margins can be cleared with **mgc**.

Shadows, Italics, Wide Characters

Five new sets of strings describe the capabilities printers have of enhancing printed text.

Enhanced Printing	
sshm	Enter shadow-printing mode
rshm	Exit shadow-printing mode
sitm	Enter italicizing mode
ritm	Exit italicizing mode
swidm	Enter wide character mode
rwidm	Exit wide character mode
ssupm	Enter superscript mode
rsupm	Exit superscript mode
supcs	List of characters available as superscripts
ssubm	Enter subscript mode
rsubm	Exit subscript mode
subcs	List of characters available as subscripts

If a printer requires the **sshm** control sequence before every character to be shadow-printed, the **rshm** string is left blank. Thus programs that find a control sequence in **sshm** but none in **rshm** should use the **sshm** control sequence before every character to be shadow-printed; otherwise, the **sshm** control sequence should be used once before the set of characters to be shadow-printed, followed by **rshm**. The same is also true of each of the **sitm**/**ritm**, **swidm**/**rwidm**, **ssupm**/**rsupm**, and **ssubm**/ **rsubm** pairs.

Note that **terminfo** also has a capability for printing emboldened text (**bold**). While shadow printing and emboldened printing are similar in that they ''darken'' the text, many printers produce these two types of print in slightly different ways. Generally, emboldened printing is done by overstriking the same character one or more times. Shadow printing likewise usually involves overstriking, but with a slight movement up and/or to the side so that the character is ''fatter.''

It is assumed that enhanced printing modes are independent modes, so that it would be possible, for instance, to shadow print italicized subscripts.

As mentioned earlier, the amount of motion automatically made after printing a wide character should be given in **widcs**.

If only a subset of the printable ASCII characters can be printed as superscripts or subscripts, they should be listed in **supcs** or **subcs** strings, respectively. If the **ssupm** or **ssubm** strings contain control sequences, but the corresponding **supcs** or **subcs** strings

are empty, it is assumed that all printable ASCII characters are available as superscripts or subscripts.

Automatic motion made after printing a superscript or subscript is assumed to be the same as for regular characters. Thus, for example, printing any of the following three examples will result in equivalent motion:

Bi $\mathbf{B_i}$ $\mathbf{B^i}$

Note that the existing **msgr** boolean capability describes whether motion control sequences can be used while in ''standout mode.'' This capability is extended to cover the enhanced printing modes added here. **msgr** should be set for those printers that accept any motion control sequences without affecting shadow, italicized, widened, superscript, or subscript printing. Conversely, if **msgr** is not set, a program should end these modes before attempting any motion.

Section 2-5: Alternate Character Sets

In addition to allowing you to define line graphics (described in Section 1-12), **terminfo** lets you define alternate character sets. The following capabilities cover printers and terminals with multiple selectable or definable character sets.

Alternate Character Sets

scs	Select character set *N*
scsd	Start definition of character set *N*, *M* characters
defc	Define character *A*, *B* dots wide, descender *D*
rcsd	End definition of character set *N*
csnm	List of character set names
daisy	Printer has manually changed print-wheels

The **scs**, **rcsd**, and **csnm** strings are used with a single argument, *N*, a number from 0 to 63 that identifies the character set. The **scsd** string is also used with the argument *N* and another, *M*, that gives the number of characters in the set. The **defc** string is used with three arguments: *A* gives the ASCII code representation for the character, *B* gives the width of the character in dots, and *D* is zero or one depending on whether the character is a ''descender'' or not. The **defc** string is also followed by a string of ''image-data'' bytes that describe how the character looks (see below).

Character set 0 is the default character set present after the printer has been initialized. Not every printer has 64 character sets, of course; using **scs** with an argument that doesn't select an available character set should cause a null result from **tparm**.

If a character set has to be defined before it can be used, the **scsd** control sequence is to be used before defining the character set, and the **rcsd** is to be used after. They should also cause a null result from **tparm** when used with an argument *N* that doesn't apply. If a character set still has to be selected after being defined, the **scs** control sequence should follow the **rcsd** control sequence. By examining the results of using each of the **scs**, **scsd**,

and **rcsd** strings with a character set number in a call to **tparm**, a program can determine which of the three are needed.

Between use of the **scsd** and **rcsd** strings, the **defc** string should be used to define each character. To print any character on printers covered by **terminfo**, the ASCII code is sent to the printer. This is true for characters in an alternate set as well as ''normal'' characters. Thus the definition of a character includes the ASCII code that represents it. In addition, the width of the character in dots is given, along with an indication of whether the character should descend below the print line (such as the lower case letter ''g'' in most character sets). The width of the character in dots also indicates the number of image-data bytes that will follow the **defc** string. These image-data bytes indicate where in a dot-matrix pattern ink should be applied to ''draw'' the character; the number of these bytes and their form are defined below under ''Dot-Mapped Graphics.''

It's easiest for the creator of **terminfo** entries to refer to each character set by number; however, these numbers will be meaningless to the application developer. The **csnm** string alleviates this problem by providing names for each number.

When used with a character set number in a call to **tparm**, the **csnm** string will produce the equivalent name. These names should be used as a reference only. No naming convention is implied, although anyone who creates a **terminfo** entry for a printer should use names consistent with the names found in user documents for the printer. Application developers should allow a user to specify a character set by number (leaving it up to the user to examine the **csnm** string to determine the correct number), or by name, where the application examines the **csnm** string to determine the corresponding character set number.

These capabilities are likely to be used only with dot-matrix printers. If they are not available, the strings should not be defined. For printers that have manually changed print-wheels or font cartridges, the boolean **daisy** is set.

Section 2-6: Dot-Matrix Graphics

Dot-matrix printers typically have the capability of reproducing ''raster-graphics'' images. Three new numeric capabilities and three new string capabilities can help a program draw raster-graphics images independent of the type of dot-matrix printer or the number of pins or dots the printer can handle at one time.

Dot-Matrix Graphics

npins	Number of pins, *N*, in print-head
spinv	Spacing of pins vertically in pins per inch
spinh	Spacing of dots horizontally in dots per inch
porder	Matches software bits to print-head pins
sbim	Start printing bit image graphics, *B* bits wide
rbim	End printing bit image graphics

The **sbim** sring is used with a single argument, *B*, the width of the image in dots.

The model of dot-matrix or raster-graphics that **terminfo** presents is similar to the technique used for most dot-matrix printers: each pass of the printer's print-head is assumed to produce a dot-matrix that is *N* dots high and *B* dots wide. This is typically a wide, squat, rectangle of dots. The height of this rectangle in dots will vary from one printer to the next; this is given in the **npins** numeric capability. The size of the rectangle in fractions of an inch will also vary; it can be deduced from the **spinv** and **spinh** numeric capabilities.
With these three values an application can divide a complete raster-graphics image into several horizontal strips, perhaps interpolating to account for different dot spacing vertically and horizontally.

The **sbim** and **rbim** strings are used to start and end a dot-matrix image, respectively. The **sbim** string is used with a single argument that gives the width of the dot-matrix in dots. A sequence of ''image-data bytes'' are sent to the printer after the **sbim** string and before the **rbim** string. The number of bytes is a integral multiple of the width of the dot-matrix; the multiple and the form of each byte is determined by the **porder** string as described below.

The **porder** string is a comma separated list of pin numbers optionally followed by an numerical offset. The offset, if given, is separated from the list with a semicolon. The position of each pin number in the list corresponds to a bit in an 8-bit data byte. The pins are numbered consecutively from 1 to **npins**, with 1 being the top pin. Note that the term ''pin'' is used loosely here; ''ink-jet'' dot-matrix printers don't have pins, but can be considered to have an equivalent method of applying a single dot of ink to paper. The bit positions in **porder** are in groups of 8, with the first position in each group the most significant bit and the last position the least significant bit. An application produces 8-bit bytes in the order of the groups in **porder**.

An application computes the ''image-data bytes'' from the internal image, mapping vertical dot positions in each print-head pass into 8-bit bytes, using a 1 bit where ink should be applied and 0 where no ink should be applied. This can be reversed (0 bit for ink, 1 bit for no ink) by giving a negative pin number. If a position is skipped in **porder**, a 0 bit is used. If a position has a lower case 'x' instead of a pin number, a 1 bit is used in the skipped position. For consistency, a lower case 'o' can be used to represent a 0 filled, skipped bit. There must be a multiple of 8 bit positions used or skipped in **porder**; if not, 0 bits are used to fill the last byte in the least significant bits. The offset, if given, is added to each data byte; the offset can be negative.

Some examples may help clarify the use of the **porder** string. The AT&T 470, AT&T 475 and C.Itoh 8510 printers provide eight pins for graphics. The pins are identified top to bottom by the 8 bits in a byte, from least significant to most. The **porder** strings for these printers would be **8,7,6,5,4,3,2,1**. The AT&T 478 and AT&T 479 printers also provide eight pins for graphics. However, the pins are identified in the reverse order. The **porder**

strings for these printers would be **1,2,3,4,5,6,7,8**. The AT&T 5310, AT&T 5320, DEC LA100, and DEC LN03 printers provide six pins for graphics. The pins are identified top to bottom by the decimal values 1, 2, 4, 8, 16 and 32. These correspond to the low six bits in an 8-bit byte, although the decimal values are further offset by the value 63. The **porder** string for these printers would be **„6,5,4,3,2,1;63**, or alternately **o,o,6,5,4,3,2,1;63**.

Section 2-7: Effect of Changing Printing Resolution

If the control sequences to change the character pitch or the line pitch are used, the pin or dot spacing may change:

Dot-Matrix Graphics
Changing the Character / Line Pitches

cpi	Change character pitch
cpix	If set, **cpi** changes **spinh**
lpi	Change line pitch
lpix	If set, **lpi** changes **spinv**

Programs that use **cpi** or **lpi** should recalculate the dot spacing:

Dot-Matrix Graphics
Effects of Changing the Character / Line Pitches

Before	*After*
Using **cpi** with **cpix** clear:	
$bold spinh ’$	$bold spinh$
Using **cpi** with **cpix** set:	
$bold spinh ’$	$bold spinh = bold spinh ’ cdot bold orhi over { bold {orhi ’} }$
Using **lpi** with **lpix** clear:	
$bold spinv ’$	$bold spinv$
Using **lpi** with **lpix** set:	
$bold spinv ’$	$bold spinv = bold {spinv ’} cdot bold orhi over { bold {orhi ’}}$
Using **chr:**	
$bold spinh ’$	$bold spinh$
Using **cvr:**	
$bold spinv ’$	$bold spinv$

orhi’ and **orhi** are the values of the horizontal resolution in steps per inch, before using **cpi** and after using **cpi**, respectively. Likewise, **orvi’** and **orvi** are the values of the vertical resolution in steps per inch, before using **lpi** and after using **lpi**, respectively. Thus, the changes in the dots per inch for dot-matrix graphics follow the changes in steps per inch for printer resolution.

Section 2-8: Print Quality

Many dot-matrix printers can alter the dot spacing of printed text to produce near ''letter quality'' printing or ''draft quality'' printing. Usually it is important to be able to choose one or the other because the rate of printing generally falls off as the quality improves. There are three new strings used to describe these capabilities.

Print Quality	
snlq	Set near-letter quality print
snrmq	Set normal quality print
sdrfq	Set draft quality print

The capabilities are listed in decreasing levels of quality. If a printer doesn't have all three levels, one or two of the strings should be left blank as appropriate.

Section 2-9: Printing Rate and Buffer Size

Because there is no standard protocol that can be used to keep a program synchronized with a printer, and because modern printers can buffer data before printing it, a program generally cannot determine at any time what has been printed. Two new numeric capabilities can help a program estimate what has been printed.

Print Rate/Buffer Size	
cps	Nominal print rate in characters per second
bufsz	Buffer capacity in characters

cps is the nominal or average rate at which the printer prints characters; if this value is not given, the rate should be estimated at one-tenth the prevailing baud rate. **bufsz** is the maximum number of subsequent characters buffered before the guaranteed printing of an earlier character, assuming proper flow control has been used. If this value is not given it is assumed that the printer does not buffer characters, but prints them as they are received.

As an example, if a printer has a 1000-character buffer, then sending the letter ''a'' followed by 1000 additional characters is guaranteed to cause the letter ''a'' to print. If the same printer prints at the rate of 100 characters per second, then it should take 10 seconds to print all the characters in the buffer, less if the buffer is not full. By keeping track of the characters sent to a printer, and knowing the print rate and buffer size, a program can synchronize itself with the printer.

Note that most printer manufacturers advertise the maximum print rate, not the nominal print rate. A good way to get a value to put in for **cps** is to generate a few pages of text, count the number of printable characters, and then see how long it takes to print the text.

Applications that use these values should recognize the variability in the print rate. Straight text, in short lines, with no embedded control sequences will probably print at close to the advertised print rate and probably faster than the rate in **cps**. Graphics data with a lot of control sequences, or very long lines of text, will print at well below the advertised rate and below the rate in **cps**. If the application is using **cps** to decide how

long it should take a printer to print a block of text, the application should pad the estimate. If the application is using **cps** to decide how much text has already been printed, it should shrink the estimate. The application will thus err in favor of the user, who wants, above all, to see all the output in its correct place.

FILES

/usr/share/lib/terminfo/?/*	compiled terminal description database
/usr/share/lib/.COREterm/?/*	subset of compiled terminal description database
/usr/share/lib/tabset/*	tab settings for some terminals, in a format appropriate to be output to the terminal (escape sequences that set margins and tabs)

SEE ALSO

ls(1), **pg**(1), **stty**(1), **tput**(1), **tty**(1), **vi**(1), **tic**(1M), **printf**(3S)

NOTES

The most effective way to prepare a terminal description is by imitating the description of a similar terminal in **terminfo** and to build up a description gradually, using partial descriptions with a screen oriented editor, such as **vi**, to check that they are correct. To easily test a new terminal description the environment variable **TERMINFO** can be set to the pathname of a directory containing the compiled description, and programs will look there rather than in **/usr/share/lib/terminfo**.

NAME

timezone – default timezone data base

SYNOPSIS

/etc/timezone

DESCRIPTION

The timezone file contains information regarding the default timezone for each host in a domain. Alternatively, a single default line for the entire domain may be specified. Each entry has the format:

Timezone-name official-host-or-domain-name

Items are separated by any number of blanks and/or TAB characters. A '#' indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file. The timezone is a pathname relative to the directory **/usr/share/lib/zoneinfo**.

This file is not actually referenced by any system software; it is merely used as a source file to construct the NIS **timezone.byname** map. This map is read by the program **/usr/etc/install/sysIDtool** to initialize the timezone of the client system at installation time.

The **timzone** file does not set the timezone environment variable **TZ**. See **TIMEZONE**(4) for information to set the **TZ** environment variable.

EXAMPLES

Here is a typical line from the **/etc/timezone** file:

US/Eastern East.Sun.COM #Sun East Coast

FILES

/etc/timezone

SEE ALSO

TIMEZONE(4)

NAME

ts_dptbl – time-sharing dispatcher parameter table

DESCRIPTION

The process scheduler (or dispatcher) is the portion of the kernel that controls allocation of the CPU to processes. The scheduler supports the notion of scheduling classes where each class defines a scheduling policy, used to schedule processes within that class. Associated with each scheduling class is a set of priority queues on which ready to run processes are linked. These priority queues are mapped by the system configuration into a set of global scheduling priorities which are available to processes within the class. (The dispatcher always selects for execution the process with the highest global scheduling priority in the system.) The priority queues associated with a given class are viewed by that class as a contiguous set of priority levels numbered from 0 (lowest priority) to *n* (highest priority—a configuration-dependent value). The set of global scheduling priorities that the queues for a given class are mapped into might not start at zero and might not be contiguous (depending on the configuration).

Processes in the time-sharing class which are running in user mode (or in kernel mode before going to sleep) are scheduled according to the parameters in a time-sharing dispatcher parameter table (**ts_dptbl**). Processes in the inter-active scheduling class are also scheduled according to the parameters in the time-sharing dispatcher parameter table. (Time-sharing processes and inter-active processes running in kernel mode after sleeping are run within a special range of priorities reserved for such processes and are not affected by the parameters in the **ts_dptbl** until they return to user mode.) The **ts_dptbl** consists of an array (**config_ts_dptbl[]**) of parameter structures (**struct tsdpent_t**), one for each of the *n* priority levels used by time-sharing processes and inter-active processes in user mode. The structures are accessed via a pointer, (**ts_dptbl**), to the array. The properties of a given priority level *i* are specified by the *i*th parameter structure in this array (**ts_dptbl[** *i***]**).

A parameter structure consists of the following members. These are also described in the **/usr/include/sys/ts.h** header.

ts_globpri The global scheduling priority associated with this priority level. The mapping between time-sharing priority levels and global scheduling priorities is determined at boot time by the system configuration. **ts_globpri** is the only member of the **ts_dptbl** which cannot be changed with **dispadmin**(1M).

ts_quantum The length of the time quantum allocated to processes at this level in ticks (Hz).

ts_tqexp Priority level of the new queue on which to place a process running at the current level if it exceeds its time quantum. Normally this field links to a lower priority time-sharing level that has a larger quantum.

ts_slpret Priority level of the new queue on which to place a process, that was previously in user mode at this level, when it returns to user mode after sleeping. Normally this field links to a higher priority level that has a smaller quantum.

ts_maxwait A per process counter, **ts_dispwait** is initialized to zero each time a time-sharing or inter-active process is placed back on the dispatcher queue after its time quantum has expired or when it is awakened (**ts_dispwait** is not reset to zero when a process is preempted by a higher priority process). This counter is incremented once per second for each process on the dispatcher queue. If a process's **ts_dispwait** value exceeds the **ts_maxwait** value for its level, the process's priority is changed to that indicated by **ts_lwait**. The purpose of this field is to prevent starvation.

ts_lwait Move a process to this new priority level if **ts_dispwait** is greater than **ts_maxwait**.

An administrator can affect the behavior of the time-sharing portion of the scheduler by reconfiguring the **ts_dptbl**. Since processes in the time-sharing and inter-active scheduling classes share the same dispatch parameter table (**ts_dptbl**), changes to this table will affect both scheduling classes. There are two methods available for doing this: reconfigure with a loadable module at boot-time or by using **dispadmin**(1M) at run-time.

TS_DPTBL LOADABLE MODULE

The **ts_dptbl** can be reconfigured with a loadable module which contains a new time sharing dispatch table. The module containing the dispatch table is separate from the TS loadable module which contains the rest of the time-sharing and inter-active software. This is the only method that can be used to change the number of time-sharing priority levels or the set of global scheduling priorities used by the time-sharing and inter-active classes. The relevant procedure and source code is described in the **REPLACING THE TS_DPTBL LOADABLE MODULE** section.

DISPADMIN CONFIGURATION FILE

With the exception of **ts_globpri** all of the members of the **ts_dptbl** can be examined and modified on a running system using the **dispadmin**(1M) command. Invoking **dispadmin** for the time-sharing or inter-active class allows the administrator to retrieve the current **ts_dptbl** configuration from the kernel's in-core table, or overwrite the in-core table with values from a configuration file. The configuration file used for input to **dispadmin** must conform to the specific format described below.

Blank lines are ignored and any part of a line to the right of a # symbol is treated as a comment. The first non-blank, non-comment line must indicate the resolution to be used for interpreting the **ts_quantum** time quantum values. The resolution is specified as

RES=*res*

where *res* is a positive integer between 1 and 1,000,000,000 inclusive and the resolution used is the reciprocal of *res* in seconds (for example, **RES=1000** specifies millisecond resolution). Although very fine (nanosecond) resolution may be specified, the time quantum lengths are rounded up to the next integral multiple of the system clock's resolution.

The remaining lines in the file are used to specify the parameter values for each of the time-sharing priority levels. The first line specifies the parameters for time-sharing level 0, the second line specifies the parameters for time-sharing level 1, etc. There must be exactly one line for each configured time-sharing priority level.

EXAMPLES

The following excerpt from a **dispadmin** configuration file illustrates the format. Note that for each line specifying a set of parameters there is a comment indicating the corresponding priority level. These level numbers indicate priority within the time-sharing and inter-active classes, and the mapping between these time-sharing priorities and the corresponding global scheduling priorities is determined by the configuration specified in the **ts** master file. The level numbers are strictly for the convenience of the administrator reading the file and, as with any comment, they are ignored by **dispadmin**. **dispadmin** assumes that the lines in the file are ordered by consecutive, increasing priority level (from 0 to the maximum configured time-sharing priority). The level numbers in the comments should normally agree with this ordering; if for some reason they don't, however, **dispadmin** is unaffected.

 # Time-Sharing Dispatcher Configuration File RES=1000

# ts_quantum	ts_tqexp	ts_slpret	ts_maxwait	ts_lwait	PRIORITY LEVEL
500	0	10	5	10	# 0
500	0	11	5	11	# 1
500	1	12	5	12	# 2
500	1	13	5	13	# 3
500	2	14	5	14	# 4
500	2	15	5	15	# 5
450	3	16	5	16	# 6
450	3	17	5	17	# 7
.	.	.	.	.	. .
.	.	.	.	.	. .
.	.	.	.	.	. .
50	48	59	5	59	# 58
50	49	59	5	59	# 59

REPLACING THE TS_DPTBL LOADABLE MODULE

In order to change the size of the time sharing dispatch table, the loadable module which contains the dispatch table information will have to be built. It is recommended that you save the existing module before using the following procedure.

1. Place the dispatch table code shown below in a file called **ts_dptbl.c** An example of this file follows.
2. Compile the code using the given compilation and link lines supplied.
   cc –c –0 –D_KERNEL ts_dptbl.c
   ld –r –o TS_DPTBL ts_dptbl.o
3. Copy the current dispatch table in **/kernel/sched** to **TS_DPTBL.bak**.
4. Replace the current **TS_DPTBL** in **/kernel/sched**.

5. You will have to make changes in the **/etc/system** file to reflect the changes to the sizes of the tables. See **system**(4). The two variables affected are **ts_maxupri** and **ts_maxkmdpri**. The syntax for setting these is as follows:

   set TS:ts_maxupri=(value for max time-sharing user priority)
   set TS:ts_maxkmdpri=(number of kernel mode priorities - 1)

6. Reboot the system to use the new dispatch table.

NOTE: Great care should be used in replacing the dispatch table using this method. If you do not get it right, panics may result, thus making the system unusable.

The following is an example of a **ts_dptbl.c** file used for building the new **ts_dptbl**.

```
        /* BEGIN ts_dptbl.c */

#include <sys/proc.h>
#include <sys/priocntl.h>
#include <sys/class.h>
#include <sys/disp.h>
#include <sys/ts.h>
#include <sys/rtpriocntl.h>

/*
 * This is the loadable module wrapper.
 */
#include <sys/modctl.h>

extern struct mod_ops mod_miscops;

/*
 * Module linkage information for the kernel.
 */

static struct modlmisc modlmisc = {
        &mod_miscops, "Time sharing dispatch table"
};

static struct modlinkage modlinkage = {
        MODREV_1, &modlmisc, 0
};

_init()
{
        return (mod_install(&modlinkage));
}
```

```
_info(modinfop)
        struct modinfo *modinfop;
{
        return (mod_info(&modlinkage, modinfop));
}


/*
 * array of global priorities used by ts procs sleeping or
 * running in kernel mode after sleep. Must have at least
 * 40 values.
 */

pri_t config_ts_kmdpris[] = {
        60,61,62,63,64,65,66,67,68,69,
        70,71,72,73,74,75,76,77,78,79,
        80,81,82,83,84,85,86,87,88,89,
        90,91,92,93,94,95,96,97,98,99,
};

tsdpent_t               config_ts_dptbl[] = {

/*  glbpri      qntm      tqexp     slprt     mxwt     lwt  */
     0,         100,        0,       10,       5,       10,
     1,         100,        0,       11,       5,       11,
     2,         100,        1,       12,       5,       12,
     3,         100,        1,       13,       5,       13,
     4,         100,        2,       14,       5,       14,
     5,         100,        2,       15,       5,       15,
     6,         100,        3,       16,       5,       16,
     7,         100,        3,       17,       5,       17,
     8,         100,        4,       18,       5,       18,
     9,         100,        4,       19,       5,       19,
    10,          80,        5,       20,       5,       20,
    11,          80,        5,       21,       5,       21,
    12,          80,        6,       22,       5,       22,
    13,          80,        6,       23,       5,       23,
    14,          80,        7,       24,       5,       24,
    15,          80,        7,       25,       5,       25,
    16,          80,        8,       26,       5,       26,
    17,          80,        8,       27,       5,       27,
    18,          80,        9,       28,       5,       28,
    19,          80,        9,       29,       5,       29,
    20,          60,       10,       30,       5,       30,
    21,          60,       11,       31,       5,       31,
```

```
    22,     60,     12,     32,     5,      32,
    23,     60,     13,     33,     5,      33,
    24,     60,     14,     34,     5,      34,
    25,     60,     15,     35,     5,      35,
    26,     60,     16,     36,     5,      36,
    27,     60,     17,     37,     5,      37,
    28,     60,     18,     38,     5,      38,
    29,     60,     19,     39,     5,      39,
    30,     40,     20,     40,     5,      40,
    31,     40,     21,     41,     5,      41,
    32,     40,     22,     42,     5,      42,
    33,     40,     23,     43,     5,      43,
    34,     40,     24,     44,     5,      44,
    35,     40,     25,     45,     5,      45,
    36,     40,     26,     46,     5,      46,
    37,     40,     27,     47,     5,      47,
    38,     40,     28,     48,     5,      48,
    39,     40,     29,     49,     5,      49,
    40,     20,     30,     50,     5,      50,
    41,     20,     31,     50,     5,      50,
    42,     20,     32,     51,     5,      51,
    43,     20,     33,     51,     5,      51,
    44,     20,     34,     52,     5,      52,
    45,     20,     35,     52,     5,      52,
    46,     20,     36,     53,     5,      53,
    47,     20,     37,     53,     5,      53,
    48,     20,     38,     54,     5,      54,
    49,     20,     39,     54,     5,      54,
    50,     10,     40,     55,     5,      55,
    51,     10,     41,     55,     5,      55,
    52,     10,     42,     56,     5,      56,
    53,     10,     43,     56,     5,      56,
    54,     10,     44,     57,     5,      57,
    55,     10,     45,     57,     5,      57,
    56,     10,     46,     58,     5,      58,
    57,     10,     47,     58,     5,      58,
    58,     10,     48,     59,     5,      59,
    59,     10,     49,     59,     5,      59,
};

short config_ts_maxumdpri = sizeof (config_ts_dptbl)/16 - 1;

/*
 * Return the address of config_ts_dptbl
 */
```

```
tsdpent_t *
ts_getdptbl()
{
        return (config_ts_dptbl);
}

/*
 * Return the address of config_ts_kmdpris
 */
int *
ts_getkmdpris()
{
        return (config_ts_kmdpris);
}

/*
 * Return the address of ts_maxumdpri
 */
short
ts_getmaxumdpri()
{
        return (config_ts_maxumdpri);
}

/* END ts_dptbl.c */
```

FILES

<sys/ts.h>

SEE ALSO

priocntl(1), **dispadmin**(1M), **priocntl**(2), **system**(4)

File System Administration
System Services Guide

NOTES

dispadmin does some limited sanity checking on the values supplied in the configuration file. The sanity checking is intended to ensure that the new **ts_dptbl** values do not cause the system to panic. The sanity checking does not attempt to analyze the effect that the new values will have on the performance of the system. Unusual **ts_dptbl** configurations may have a dramatic negative impact on the performance of the system.

No sanity checking is done on the **ts_dptbl** values specified in the **TS_DPTBL** loadable module. Specifying an inconsistent or nonsensical **ts_dptbl** configuration through the **TS_DPTBL** loadable module could cause serious performance problems and/or cause the system to panic.

NAME

ttydefs – file contains terminal line settings information for ttymon

DESCRIPTION

/etc/ttydefs is an administrative file that contains records divided into fields by colons (":"). This information used by **ttymon** to set up the speed and terminal settings for a TTY port.

The **ttydefs** file contains the following fields:

ttylabel The string **ttymon** tries to match against the TTY port's *ttylabel* field in the port monitor administrative file. It often describes the speed at which the terminal is supposed to run, for example, **1200**.

initial-flags Contains the initial **termio**(7) settings to which the terminal is to be set. For example, the system administrator will be able to specify what the default erase and kill characters will be. *initial-flags* must be specified in the syntax recognized by the **stty** command.

final-flags *final-flags* must be specified in the same format as *initial-flags*. **ttymon** sets these final settings after a connection request has been made and immediately prior to invoking a port's service.

autobaud If the autobaud field contains the character 'A', autobaud will be enabled. Otherwise, autobaud will be disabled. **ttymon** determines what line speed to set the TTY port to by analyzing the carriage returns entered. If autobaud has been disabled, the hunt sequence is used for baud rate determination.

nextlabel If the user indicates that the current terminal setting is not appropriate by sending a BREAK, **ttymon** searchs for a **ttydefs** entry whose *ttylabel* field matches the *nextlabel* field. If a match is found, **ttymon** uses that field as its *ttylabel* field. A series of speeds is often linked together in this way into a closed set called a hunt sequence. For example, **4800** may be linked to **1200**, which in turn is linked to **2400**, which is finally linked to **4800**.

SEE ALSO

sttydefs(1M), **ttymon**(1M)

File System Administration

NAME

ttysrch – directory search list for ttyname

DESCRIPTION

ttysrch is an optional file that is used by the **ttyname** library routine. This file contains the names of directories in **/dev** that contain terminal and terminal-related device files. The purpose of this file is to improve the performance of **ttyname** by indicating which subdirectories in **/dev** contain terminal-related device files and should be searched first. These subdirectory names must appear on separate lines and must begin with **/dev**. Those path names that do not begin with **/dev** will be ignored and a warning will be sent to the console. Blank lines (lines containing only white space) and lines beginning with the comment character "#" will be ignored. For each file listed (except for the special entry **/dev**), **ttyname** will recursively search through subdirectories looking for a match. If **/dev** appears in the **ttysrch** file, the **/dev** directory itself will be searched but there will not be a recursive search through its subdirectories.

When **ttyname** searches through the device files, it tries to find a file whose major / minor device number, file system identifier, and inode number match that of the file descriptor it was given as an argument. If a match is not found, it will settle for a match of just major / minor device and file system identifier, if one can be found. However, if the file descriptor is associated with a cloned device, this algorithm does not work efficiently because the inode number of the device file associated with a clonable device will never match the inode number of the file descriptor that was returned by the open of that clonable device. To help with these situations, entries can be put into the **/etc/ttysrch** file to improve performance when cloned devices are used as terminals on a system (for example, for remote login). However, this is only useful if the minor devices related to a cloned device are put into a subdirectory. (It is important to note that device files need not exist for cloned devices and if that is the case, **ttyname** will eventually fail.) An optional second field is used in the **/etc/ttysrch** file to indicate the matching criteria. This field is separated by white space (any combination of blanks or tabs). The letter **M** means major / minor device number, **F** means file system identifier, and **I** means inode number. If this field is not specified for an entry, the default is **MFI** which means try to match on all three. For cloned devices the field should be **MF**, which indicates that it is not necessary to match on the inode number.

Without the **/etc/ttysrch** file, **ttyname** will search the **/dev** directory by first looking in the directories **/dev/term**, **/dev/pts**, and **/dev/xt**. If a system has terminal devices installed in directories other than these, it may help performance if the **ttysrch** file is created and contains that list of directories.

EXAMPLES

A sample **/etc/ttysrch** file follows:

```
/dev/term       MFI
/dev/pts        MFI
/dev/xt         MFI
/dev/slan       MF
```

This file tells **ttyname** that it should first search through those directories listed and that when searching through the **/dev/slan** directory, if a file is encountered whose major / minor devices and file system identifier match that of the file descriptor argument to **ttyname**, this device name should be considered a match.

FILES

/etc/ttysrch

SEE ALSO

ttyname(3C)

NAME

ufsdump, dumpdates – incremental dump format

SYNOPSIS

```
#include <sys/types.h>
#include <sys/inode.h>
#include <protocols/dumprestore.h>

/etc/dumpdates
```

DESCRIPTION

Tapes used by **ufsdump**(1M) and **ufsrestore**(1M) contain:

- a header record
- two groups of bit map records
- a group of records describing directories
- a group of records describing files

The format of the header record and of the first record of each description as given in the include file **<protocols/dumprestore.h>** is:

```
#define TP_BSIZE            1024
#define NTREC               10
#define HIGHDENSITYTREC     32
#define CARTRIDGETREC       63
#define TP_NINDIR           (TP_BSIZE/2)
#define TP_NINOS            (TP_NINDIR / sizeop (long))
#define LBLSIZE             16
#define NAMELEN             64

#define NFS_MAGIC           (int) 60012
#define CHECKSUM            (int) 84446

union u_data {
        char  s_addrs[TP_NINDIR];
        long  s_inos[TP_NINOS];

union u_spcl {
        char  dummy[TP_BSIZE];
        struct s_spcl {
                long            c_type;
                time_t          c_date;
                time_t          c_ddate;
                long            c_volume;
                daddr_t         c_tapea;
                ino_t           c_inumber;
                long            c_magic;
                long            c_checksum;
                struct dinode   c_dinode;
                long            c_count;
                union           u_data c_data;
                char            c_label[LBLSIZE];
```

```
                long            c_level;
                char            c_filesys[NAMELEN];
                char            c_dev[NAMELEN];
                char            c_host[NAMELEN];
                long            c_flags;
                long            c_firstrec;
                long            c_spare[32];
        } s_spcl;
} u_spcl;

#define spcl u_spcl.s_spcl
#define c_addr c_data.s_addrs
#define c_inos cdata.s_inos

#define TS_TAPE         1
#define TS_INODE        2
#define TS_ADDR         4
#define TS_BITS         3
#define TS_CLRI         6
#define TS_END          5
#define TS_EOM          7

#define DR_NEWHEADER    1
#define DR_INODEINFO    2
#define DR_REDUMP       4
#define DR_TRUELIC      8
#define DUMPOUTFMT      "%-24s %c %s"
#define DUMPINFMT       "%24s %c %[^ \n] \n"
```

The constants are described as follows:

TP_BSIZE Size of file blocks on the dump tapes. Note that **TP_BSIZE** must be a multiple of **DEV_BSIZE**.

NTREC Default number of **TP_BSIZE** byte records in a physical tape block, changeable by the **b** option to **ufsdump**(1M).

HIGHDENSITYNTREC
Default number of **TP_BSIZE** byte records in a physical tape block on 6250 BPI or higher density tapes.

CARTRIDGETREC
Default number of **TP_BSIZE** records in a physical tape block on cartridge tapes.

TP_NINDIR Number of indirect pointers in a **TS_INODE** or **TS_ADDR** record. It must be a power of 2.

TP_NINOS The maximum number of volumes on a tape. Used for tape labeling in **hsmdump** and **hsmrestore** (available with Online:Backup 2.0 optional software package SUNWhsm).

LBLSIZE The maximum size of a volume label. Used for tape labeling in **hsmdump** and **hsmrestore** (available with Online:Backup 2.0 optional software package SUNWhsm).

NAMELEN The maximum size of a host's name.

NFS_MAGIC All header records have this number in **c_magic**.

CHECKSUM Header records checksum to this value.

The **TS_** entries are used in the **c_type** field to indicate what sort of header this is. The types and their meanings are as follows:

TS_TAPE Tape volume label.

TS_INODE A file or directory follows. The **c_dinode** field is a copy of the disk inode and contains bits telling what sort of file this is.

TS_ADDR A subrecord of a file description. See **s_addrs** below.

TS_BITS A bit map follows. This bit map has a one bit for each inode that was dumped.

TS_CLRI A bit map follows. This bit map contains a zero bit for all inodes that were empty on the file system when dumped.

TS_END End of tape record.

TS_EOM floppy EOM — restore compat with old dump

The flags are described as follows:

DR_NEWHEADER
New format tape header.

DR_INFODEINFO
Header contains starting inode info.

DR_REDUMP Dump contains recopies of active files.

DR_TRUEINC Dump is a "true incremental".

DUMPOUTFMT Name, incon, and ctime (date) for printf.

DUMPINFMT Inverse for scanf.

The fields of the header structure are as follows:

s_addrs An array of bytes describing the blocks of the dumped file. A byte is zero if the block associated with that byte was not present on the file system; otherwise, the byte is non-zero. If the block was not present on the file lsystem, no block was dumped; the block will be stored as a hole in the file. If there is not sufficient space in this record to describe all the blocks in a file, **TS_ADDR** records will be scattered through the file, each one picking up where the last left off

s_inos The starting inodes on tape.

c_type The type of the record.

c_date The date of the previous dump.

c_ddate	The date of this dump.
c_volume	The current volume number of the dump.
c_tapea	The logical block of this record.
c_inumber	The number of the inode being dumped if this is of type **TS_INODE**.
c_magic	This contains the value **MAGIC** above, truncated as needed.
c_checksum	This contains whatever value is needed to make the record sum to **CHECKSUM**.
c_dinode	This is a copy of the inode as it appears on the file system.
c_count	The count of bytes in **s_addrs**.
u_data c_data	The union of either **u_data c_data** The union of either **s_addrs** or **s_inos**.
c_label	Label for this dump.
c_level	Level of this dump.
c_filesys	Name of dumped file system.
c_dev	Name of dumped service.
c_host	Name of dumped host.
c_flags	Additional information.
c_firstrec	First record on volume.
c_spare	Reserved for future uses.

Each volume except the last ends with a tapemark (read as an end of file). The last volume ends with a **TS_END** record and then the tapemark.

The dump history is kept in the file **/etc/dumpdates**. It is an ASCII file with three fields separated by white space:

- The name of the device on which the dumped file system resides.
- The level number of the dump tape; see **ufsdump**(1M).
- The date of the incremental dump in the format generated by **ctime**(3C).

DUMPOUTFMT is the format to use when using **printf**(3S) to write an entry to **/etc/dumpdates**; **DUMPINFMT** is the format to use when using **scanf**(3S) to read an entry from **/etc/dumpdates**.

SEE ALSO **ufsdump**(1M), **ufsrestore**(1M), **ctime**(3C), **printf**(3S), **scanf**(3S), **types**(5)

NAME

unistd – header for symbolic constants

SYNOPSIS

#include <unistd.h>

DESCRIPTION

The **<unistd.h>** header defines the symbolic constants and structures which are not already defined or declared in some other header. The contents of this header are shown below.

The following symbolic constants are defined for the **access** function [see **access**(2)]:

R_OK	Test for read permission
W_OK	Test for write permission
X_OK	Test for execute (search) permission
F_OK	Test for existence of file

The constants **F_OK**, **R_OK**, **W_OK**, and **X_OK**, and the expressions **R_OK | W_OK**, **R_OK | X_OK**, and **R_OK | W_OK | X_OK** all have distinct values.

Declares the constant

NULL	null pointer

The following symbolic constants are defined for the **lockf** function [see **lockf**(3C)]:

F_ULOCK	Unlock a previously locked region
F_LOCK	Lock a region for exclusive use
F_TLOCK	Test and lock a region for exclusive use
F_TEST	Test a region for other processes locks

The following symbolic constants are defined for the **lseek** [see **lseek**(2)] and **fcntl** [see **fcntl**(2)] functions (they have distinct values):

SEEK_SET	Set file offset to *offset*
SEEK_CUR	Set file offset to current plus *offset*
SEEK_END	Set file offset to EOF plus *offset*

The following symbolic constants are defined (with fixed values):

_POSIX_VERSION	Integer value indicating version of the POSIX standard

_XOPEN_VERSION integer value indicating version of the XPG to which system is compliant

The following symbolic constants are defined to indicate that the option is present:

_POSIX_JOB_CONTROL implementation supports job control

_POSIX_SAVED_IDS the exec functions (see **exec**(2)) **save** the effective user and group

_POSIX_VDISABLE terminal special characters defined in **<termios.h>** (see **termio**(7)) can be disabled using this character

The following symbolic constants are defined for **sysconf** (see **sysconf**(3C)):

_SC_ARG_MAX
_SC_CHILD_MAX
_SC_CLK_TCK
_SC_JOB_CONTROL
_SC_NGROUPS_MAX
_SC_OPEN_MAX
_SC_PAGESIZE
_SC_PASS_MAX
_SC_SAVED_IDS
_SC_VERSION
_SC_XOPEN_VERSION

The following symbolic constants are defined for **pathconf** (see **fpathconf**(2)):

_PC_CHOWN_RESTRICTED
_PC_LINK_MAX
_PC_MAX_CANON
_PC_MAX_INPUT
_PC_NAME_MAX
_PC_NO_TRUNC
_PC_PATH_MAX
_PC_PIPE_BUF
_PC_VDISABLE

The following symbolic constants are defined for file streams:

STDIN_FILENO File number of **stdin**. It is **0**.

STDOUT_FILENO File number of **stout**. It is **1**.

STDERR_FILENO File number of **stderr**. It is **2**.

The following pathnames are defined:

GF_PATH Pathname of the group file.

PF_PATH Pathname of the passwd file.

NOTES

The following values for constants are defined:

_POSIX_VERSION	199009L
_XOPEN_VERSION	3

SEE ALSO

access(2), **exec**(2), **fcntl**(2), **fpathconf**(2), **lseek**(2), **termios**(3), **sysconf**(3C), **group**(4), **passwd**(4), **termio**(7)

NAME

updaters – configuration file for NIS updating

SYNOPSIS

/var/yp/updaters

DESCRIPTION

The file **/var/yp/updaters** is a makefile (see **make**(1S)) which is used for updating NIS databases. Databases can only be updated in a secure network (one that has a **publickey**(4) database). Each entry in the file is a make target for a particular NIS database. For example, if there is a NIS database named **publickey.byname** that can be updated, there should be a **make** target named **publickey.byname** in the **updaters** file with the command to update the file.

The information necessary to make the update is passed to the update command through standard input. The information passed is described below (all items are followed by a NEWLINE, except for the actual bytes of key and actual bytes of date).

- Network name of client wishing to make the update (a string)
- Kind of update (an integer)
- Number of bytes in key (an integer)
- Actual bytes of key
- Number of bytes in data (an integer)
- Actual bytes of data

After getting this information through standard input, the command to update the particular database should decide whether the user is allowed to make the change. If not, it should exit with the status **YPERR_ACCESS**. If the user is allowed to make the change, the command should make the change and exit with a status of zero. If there are any errors that may prevent the updater from making the change, it should exit with the status that matches a valid NIS error code described in **<rpcsvc/ypclnt.h>**.

FILES

/var/yp/updaters

SEE ALSO

make(1S), **publickey**(4)

NAME

utmp, wtmp – utmp and wtmp entry formats

SYNOPSIS

```
#include <utmp.h>
```

DESCRIPTION

utmp and **wtmp** hold user and accounting information for commands such as **who**, **write**, and **login**. These files have the following structure, defined in **<utmp.h>**:

```
#define   UTMP_FILE                 "/var/adm/utmp"
#define   WTMP_FILE                 "/var/adm/wtmp"
#define   ut_name                   ut_user

struct utmp {
        char  ut_user[8];           /* user login name */
        char  ut_id[4];             /* /sbin/inittab id (created by */
                                    /* process that puts entry in utmp) */
        char  ut_line[12];          /* device name (console, lnxx) */
        short ut_pid;               /* process id */
        short ut_type;              /* type of entry */
        struct exit_status {
              short e_termination;  /* process termination status */
              short e_exit;         /* process exit status */
        } ut_exit;                  /* exit status of a process
                                    /* marked as DEAD_PROCESS */
        time_t ut_time;             /* time entry was made */
};
/* Definitions for ut_type */
#define   EMPTY         0
#define   RUN_LVL       1
#define   BOOT_TIME     2
#define   OLD_TIME      3
#define   NEW_TIME      4
#define   INIT_PROCESS          5/* process spawned by "init" */
#define   LOGIN_PROCESS         6/* a "getty" process waiting for login */
#define   USER_PROCESS          7/* a user process */
#define   DEAD_PROCESS          8
#define   ACCOUNTING9
#define   UTMAXTYPE   ACCOUNTING/* max legal value of ut_type */
/* Below are special strings or formats used in the "ut_line" */
/* field when accounting for something other than a process. */
/* No string for the ut_line field can be more than 11 chars + */
/* a null character in length. */
#define   RUNLVL_MSG "run–level %c"
#define   BOOT_MSG    "system boot"
#define   OTIME_MSG   "old time"
#define   NTIME_MSG   "new time"
```

FILES **/var/adm/utmp**
/var/adm/wtmp

SEE ALSO **login**(1), **who**(1), **write**(1)

NAME

utmpx, wtmpx – utmpx and wtmpx entry formats

SYNOPSIS

```
#include <utmpx.h>
```

DESCRIPTION

utmpx(4) is an extended version of **utmp**(4)**.**

utmpx and **wtmpx** hold user and accounting information for commands such as **who**, **write**, and **login**. These files have the following structure as defined by **<utmpx.h>**:

```
#define    UTMPX_FILE                  "/var/adm/utmpx"
#define    WTMPX_FILE                  "/var/adm/wtmpx"
#define    ut_name                     ut_user
#define    ut_xtime                    ut_tv.tv_sec
struct utmpx {
    char    ut_user[32];               /* user login name */
    char    ut_id[4];                  /* inittab id */
    char    ut_line[32];               /* device name */
                                       /* (console, lnxx) */
    pid_t   ut_pid;                    /* process id */
    short   ut_type;                   /* type of entry */
    struct  exit_status ut_exit;       /* process termination/exit */
                                       /* status */
    struct  timeval ut_tv;             /* time entry was made */
    long    ut_session;                /* session ID, used for */
                                       /* windowing */
    long    pad[5];                    /* reserved for future use */
    short   ut_syslen;                 /* significant length of */
                                       /* ut_host */
                                       /* including terminating null */
    char    ut_host[257];              /* remote host name */
    };
/* Definitions for ut_type */
#define    EMPTY            0
#define    RUN_LVL          1
#define    BOOT_TIME        2
#define    OLD_TIME         3
#define    NEW_TIME         4
#define    INIT_PROCESS     5          /* Process spawned by "init" */
#define    LOGIN_PROCESS    6          /* A "getty" process waiting */
                                       /* for login */
#define    USER_PROCESS     7          /* A user process */
#define    DEAD_PROCESS     8
#define    ACCOUNTING       9

#define    UTMAXTYPE  ACCOUNTING      /* Largest legal value */
                                       /* of ut_type */
```

```
/* Below are special strings or formats used in the "ut_line" */
/* field when accounting for something other than a process. */
/* No string for the ut_line field can be more than 11 chars + */
/* a null character in length. */

#define   RUNLVL_MSG          "run-level %c"
#define   BOOT_MSG            "system boot"
#define   OTIME_MSG           "old time"
#define   NTIME_MSG           "new time"
#define   MOD_WIN             10
```

FILES

/var/adm/utmpx
/var/adm/wtmpx

SEE ALSO

login(1), **who**(1), **write**(1)

NAME

vfstab – table of file system defaults

DESCRIPTION

The file **/etc/vfstab** describes defaults for each file system. The information is stored in a table with the following column headings:

device to mount	*device to fsck*	*mount point*	*FS type*	*fsck pass*	*mount at boot*	*mount options*

The fields in the table are space-separated and show the resource name, the raw device to **fsck**, the default mount directory, the name of the file system type, the number used by **fsck** to decide whether to check the file system automatically, whether the file system should be mounted automatically by **mountall**, and the mount options. A '-' is used to indicate no entry in a field. This may be used when a field does not apply to the resource being mounted.

The **getvfsent**(3C) family of routines is used to read and write to **/etc/vfstab**.

/etc/vfstab may be used to specify swap areas. An entry so specified, (which can be a file or a device), will automatically be added as a swap area by the **/sbin/swapadd** script when the system boots. To specify a swap area, the *device-to-mount* field contains the name of the swap file or device, the *FS-type* is "swap", *mount-at-boot* is "no" and all other fields have no entry.

SEE ALSO

fsck(1M), **mount**(1M), **mount_cachefs**(1M), **mount_hsfs**(1M), **mount_nfs**(1M), **mount_tmpfs**(1M), **mount_ufs**(1M), **swap**(1M), **setmnt**(1M), **getvfsent**(3C)

File System Administration

NAME

vme – configuration files for VMEbus device drivers

AVAILABILITY

SPARC

DESCRIPTION

Some Solaris platforms support the VMEbus as a peripheral expansion bus to allow VME devices to be connected to the system. Drivers for these devices need to use driver configuration files to inform the system that the device hardware may be present. The configuration file also must specify the device addresses on the VMEbus and any interrupt capabilities that the device may have.

Configuration files for VMEbus device drivers should identify the parent bus driver implicitly using the *class* keyword. This removes the dependency on the name of the particular bus driver involved since this may be named differently on different platforms. See **driver.conf**(4) for further details of configuration file syntax.

All bus drivers of class **vme** recognise the following properties:

reg An arbitrary length array where each element of the array consists of a 3-tuple of integers. Each array element describes a logically contiguous mappable resource on the VMEbus.

The first integer of the tuple specifies the type of access. The value is derived from the size of transfer and the address modifier bits used to access the locations. The table below shows the values used for common VME devices accessed in supervisor mode:

Address space	Value
A16D16	**0x2d**
A24D16	**0x3d**
A32D16	**0xd**
A16D32	**0x6d**
A24D32	**0x7d**
A32D32	**0x4d**

The second integer of each 3-tuple specifies the offset in the address space identified by the first element. The third integer of each 3-tuple specifies the size, in bytes, of the mappable region.

The driver can refer to the elements of this array by index, and construct kernel mappings to these addresses using **ddi_map_regs**(9F). The index into the array is passed as the *rnumber* argument of **ddi_map_regs( )**.

interrupts An arbitrary length array where each element of the array consists of a pair of integers. Each array element describes a possible interrupt that the device might generate.

The first integer of each pair specifies the VMEbus interrupt level. The second integer of each pair specifies the VMEbus vector number. The driver can refer to the elements of this array by index, and register interrupt handlers with the system using **ddi_add_intr**(9F). The index into

the array is passed as the *inumber* argument of **ddi_add_intr( )**.

All VMEbus device drivers must provide **reg** properties. The first two integer elements of this property are used to construct the address part of the device name under **/devices**.

Only devices that generate interrupts need to provide **interrupts** properties.

EXAMPLES

Here is a configuration file called **SUNW,diskctrl.conf** for a VMEbus disk controller card called **SUNW,diskctrl**.

The device provides two sets of registers, both should be accessed with supervisor accesses and the A16D32 address modifier bits (16 bits of address, 32 bit data transfers). Both registers occupy 32 bytes; one register set starts at address 0xee80, the other is at 0xef00. The device can generate interrupts at VME level 2 with a VME vector number of 0x92.

```
#
# Copyright (c) 1992, by Sun Microsystems, Inc.
#
#ident  "@(#)SUNW,diskctrl.conf 1.4     92/05/11 SMI"

name="SUNW,diskctrl" class="vme"
                reg=0x6d,0xee80,32,0x6d,0xef00,32
                interrupts=2,0x92;
```

SEE ALSO

driver.conf(4), **ddi_add_intr**(9F), **ddi_map_regs**(9F), **ddi_prop_op**(9F)

Writing Device Drivers

ANSI/IEEE Std 1014-1987: IEEE Standard for a Versatile Backplane Bus: VMEbus

NAME

vold.conf – Volume Management configuration file

SYNOPSIS

/etc/vold.conf

DESCRIPTION

The **vold.conf** file contains the Volume Management configuration information used by **vold**(1M). This information includes the database to use, labels that are supported, devices to use, actions to take when certain media events occur, and the list of file systems that are unsafe to eject without unmounting.

Modify **vold.conf** to specify which program should be called when media events happen (actions) or when you need to add another device to your system. See the example section for more information on adding devices.

If you modify **vold.conf**, you must tell **vold** to reread **vold.conf** by sending a HUP signal. Use

 # ps -ef | grep vold

 # kill -HUP *vold_pid*

File Format

The syntax for the **vold.conf** file is shown here.

 # Database to use
db *database*

 # Labels supported
label *label_type shared_object device*

 # Devices to use
use *device type special shared_object symname* [*options*]

 # Actions
insert *regex* [*options*] *program program args*
eject *regex* [*options*] *program program args*
notify *regex* [*options*] *program program args*

 # List of file system types unsafe to eject
unsafe *fs_type fs_type*

Of these syntax fields, you can safely modify **Devices to use** and **Actions**.

Devices to Use Field

All **use** *device* statements must be grouped together by device type. (For example, all **use cdrom** statements must be grouped together; and all **use floppy** statements must be grouped together.) Here are the explanations of the syntax for the **Devices to use** field.

device	The type of removable media device to be used. Legal values are **cdrom** and **floppy**.
type	The specific capabilities of the device. Legal value is **drive**.
special	This **sh**(1) expression specifies the device or devices to be used. Path usually begins with **/dev**.

shared_object	The name of the program that manages this device. **vold**(1M) expects to find this program in **/usr/lib/vold**.
symname	The symbolic name that refers to this device. The *symname* is placed in the device directory.
options	The user, group, and mode permissions for the media inserted (optional).

The *special* and *symname* parameters are related. If *special* contains any shell wildcard characters (i.e., has one or more asterisks or question marks in it), then the *syname* must have a "%d" at its end. In this case, the devices that are found to match the regular expression are sorted, then numbered. The first device will have a zero filled in for the "%d", the second device found will have a one, and so on.

If the *special* specification does not have any shell wildcard characters then the *symname* parameter must explicitly specify a number at its end (see **EXAMPLES** below).

Actions Field

Here are the explanations of the syntax for the **Actions** field.

insert | **eject** | **notify**	The media event prompting the event
regex	This **sh**(1) regular expression is matched against each entry in the **/vol** file system that is being affected by this event.
options	You can specify what user or group name that this event is to run as (optional).
program	The full path name of an executable program to be run when *regex* is matched.
program args	Arguments to the program.

Default Values

The default **vold.conf** file is shown here.

```
#
# Volume Daemon Configuration file
#

# Database to use (must be first)
db db_mem.so

# Labels supported
label dos label_dos.so floppy
label cdrom label_cdrom.so cdrom
label sun label_sun.so floppy

# Devices to use
use cdrom drive /dev/dsk/c*s2 dev_cdrom.so cdrom%d
use floppy drive /dev/diskette[0-9] dev_floppy.so floppy%d
```

```
# Actions
insert /vol*/dev/fd[0-9]/* user=root /usr/sbin/rmmount
insert /vol*/dev/dsk/* user=root /usr/sbin/rmmount
eject /vol*/dev/fd[0-9]/* user=root /usr/sbin/rmmount
eject /vol*/dev/dsk/* user=root /usr/sbin/rmmount
notify /vol*/rdsk/* group=tty user=root /usr/lib/vold/volmissing -p

# List of file system types unsafe to eject
unsafe ufs hsfs pcfs
```

EXAMPLES

To add a CD-ROM drive to the **vold.conf** file that does not match the default regular expression (**/dev/rdsk/c*s2**), you must explicitly list its device path and what symbolic name (with **%d**) you want the device path to have. For example, to add a CD-ROM drive that has the path **/dev/rdsk/my/cdrom***s?* (where *s?* are the different slices), add the following line to **vold.conf** (all on one line):

use cdrom drive /dev/rdsk/my/cdroms2 dev_cdrom.so cdrom%d

Then, when a volume is inserted in this CD-ROM drive. volume management will assign it the next symbolic name. For example, if two CD-ROMs match the default regular expression, they would be named **cdrom0** and **cdrom1**; and any that match the added regular expression would be named starting with **cdrom2**.

For a diskette that does not match the **vold.conf** default regular expression (**/dev/floppy**[0-9]), a similar line would have to be added for the diskette. For example, to add a diskette whose path was **/dev/my/fd0**, you would add the following to **vold.conf**:

use floppy drive /dev/my/fd0 dev_floppy.so floppy%d

SEE ALSO

volcancel(1), **volcheck**(1), **volmissing**(1) **rmmount**(1M), **vold**(1M), **rmmount.conf**(4), **volfs**(7)

NOTES

Volume Management manages both the block and character device for CD-ROMs and floppy disks; but, to make the configuration file easier to set up and scan, only one of these devices needs to be specified. Volume Management figures out both device names given one of them (if you specify the block device it figures out the pathname to the character device and vice-versa) provided you follow the conventions below.

CD-ROM Naming Conventions

The CD-ROM pathname must have a directory component of **rdsk** (for the character device) and **dsk** for the block device. For example, if you specify the character device using the line:

use cdrom drive /dev/rdsk/my/cdroms2 dev_cdrom.so cdrom%d

then it is assumed that the block device is at

/dev/dsk/my/cdroms2

Floppy Disk Naming Conventions

For floppy disks, Volume Management requires that the device pathnames end in either **rfd**[0-9] or **rdiskette**[0-9] for the character device, and **fd**[0-9] or **diskette**[0-9] for the block device. As with the CD-ROM, it generates either the block name given the character

name, or vice-versa.

NAME

ypfiles – Network Information Service Version 2, formerly knows as YP

DESCRIPTION

The NIS network information service uses a distributed, replicated database of **dbm** files contained in the **/var/yp** directory hierarchy on each NIS server. NIS has been replaced by NIS+, the new version of the Network Information Service. See **nis+**(1). This release only supports the client functionality of NIS, (see **ypclnt**(3N)). The client functions are either supported by the **ypserv** process running on a machine with an earlier version of SunOS or by the NIS+ server in "YP-compatibility" mode, (see **rpc.nisd**(1M)).

A **dbm** database served by the NIS server is called an NIS *map*. An NIS *domain* is a sub-directory of **/var/yp** containing a set of NIS maps on each NIS server.

FILES

/var/yp
/var/yp/nicknames

SEE ALSO

nis+(1), **nisaddent**(1M), **nissetup**(1M), **rpc.nisd**(1M), **ypbind**(1M), **ypinit**(1M), **dbm**(3B), **secure_rpc**(3N), **ypclnt**(3N)

NOTES

The NIS+ server, **rpc.nisd,** when run in "YP-compatibility mode", can support NIS clients only for the standard NIS maps listed below, provided that it has been set up to serve the corresponding NIS+ tables using **nissetup**(1M) and **nisaddent**(1M). The NIS+ server should serve the directory with the same name (case sensitive) as the domainname of the NIS client. NIS+ servers use secure RPC to verify client credentials but the NIS clients do not authenticate their requests using secure RPC. Therefore, NIS clients can look up the information stored by the NIS+ server only if the information has "read" access for an unauthenticated client (i.e. one with "nobody" NIS+ credentials).

NIS maps	*NIS+ tables*
passwd.byname	**passwd.org_dir**
passwd.byuid	**passwd.org_dir**
group.byname	**group.org_dir**
group.bygid	**group.org_dir**
publickey.byname	**cred.org_dir**
hosts.byaddr	**hosts.org_dir**
hosts.byname	**hosts.org_dir**
mail.byaddr	**mail_aliases.org_dir**
mail.aliases	**mail_aliases.org_dir**
services.byname	**services.org_dir**
services.byservicename	**services.org_dir**
rpc.bynumber	**rpc.org_dir**
rpc.byname	**rpc.org_dir**

protocols.bynumber	**protocols.org_dir**
protocols.byname	**protocols.org_dir**
networks.byaddr	**networks.org_dir**
networks.byname	**networks.org_dir**
netmasks.bymask	**netmasks.org_dir**
netmasks.byaddr	**netmasks.org_dir**
ethers.byname	**ethers.org_dir**
ethers.byaddr	**ethers.byname**
bootparams	**bootparams**
auto.master	**auto_master.org_dir**
auto.home	**auto_home.org_dir**
auto.direct	**auto_direct.org_dir**
auto.src	**auto_src.org_dir**

Index

G

H

I

K

L