man pages section 4: File Formats

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

Part No: 806-0633-10
February 2000

Copyright 2000 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, California 94303-4900 U.S.A. All rights reserved.

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

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

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

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

RESTRICTED RIGHTS: Use, duplication, or disclosure by the U.S. Government is subject to restrictions of FAR 52.227–14(g)(2)(6/87) and FAR 52.227–19(6/87), or DFAR 252.227–7015(b)(6/95) and DFAR 227.7202–3(a).

DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

Copyright 2000 Sun Microsystems, Inc. 901 San Antonio Road, Palo Alto, Californie 94303-4900 Etats-Unis. Tous droits réservés.

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

Des parties de ce produit pourront être dérivées du système Berkeley BSD licenciés par l'Université de Californie. UNIX est une marque déposée aux Etats-Unis et dans d'autres pays et licenciée exclusivement par X/Open Company, Ltd.

Sun, Sun Microsystems, le logo Sun, docs.sun.com, AnswerBook, AnswerBook2, et Solaris sont des marques de fabrique ou des marques déposées, ou marques de service, de Sun Microsystems, Inc. aux Etats-Unis et dans d'autres pays. Toutes les marques SPARC sont utilisées sous licence et sont des marques de fabrique ou des marques déposées de SPARC International, Inc. aux Etats-Unis et dans d'autres pays. Les produits portant les marques SPARC sont basés sur une architecture développée par Sun Microsystems, Inc.

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

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

Adobe PostScript

Please Recycle

Contents

Preface

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

Overview

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

- Section 1 describes, in alphabetical order, commands available with the operating system.
- Section 1M describes, in alphabetical order, commands that are used chiefly for system maintenance and administration purposes.
- Section 2 describes all of the system calls. Most of these calls have one or more error returns. An error condition is indicated by an otherwise impossible returned value.
- Section 3 describes functions found in various libraries, other than those functions that directly invoke UNIX system primitives, which are described in Section 2.
- Section 4 outlines the formats of various files. The C structure declarations for the file formats are given where applicable.
- Section 5 contains miscellaneous documentation such as character-set tables.
- Section 6 contains available games and demos.
- Section 7 describes various special files that refer to specific hardware peripherals and device drivers. STREAMS software drivers, modules and the STREAMS-generic set of system calls are also described.

- Section 9 provides reference information needed to write device drivers in the kernel environment. It describes two device driver interface specifications: the Device Driver Interface (DDI) and the Driver/Kernel Interface (DKI).
- Section 9E describes the DDI/DKI, DDI-only, and DKI-only entry-point routines a developer can include in a device driver.
- Section 9F describes the kernel functions available for use by device drivers.
- Section 9S describes the data structures used by drivers to share information between the driver and the kernel.

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

NAME

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

SYNOPSIS

This section shows the syntax of commands or functions. When a command or file does not exist in the standard path, its full path name is shown. Options and arguments are alphabetized, with single letter arguments first, and options with arguments next, unless a different argument order is required.

The following special characters are used in this section:

[ ]	Brackets. The option or argument enclosed in these brackets is optional. If the brackets are omitted, the argument must be specified.
. . .	Ellipses. Several values can be provided for the previous argument, or the previous argument can be specified multiple times, for example, "`filename` . . ." .
|	Separator. Only one of the arguments separated by this character can be specified at a time.
{ }	Braces. The options and/or arguments enclosed within braces are

interdependent, such that everything enclosed must be treated as a unit.

PROTOCOL — This section occurs only in subsection 3R to indicate the protocol description file.

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

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

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

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

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

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

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

USAGE

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

- Commands
- Modifiers
- Variables
- Expressions
- Input Grammar

EXAMPLES

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

ENVIRONMENT VARIABLES

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

EXIT STATUS

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

FILES

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

ATTRIBUTES

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

SEE ALSO	This section lists references to other man pages, in-house documentation, and outside publications.
DIAGNOSTICS	This section lists diagnostic messages with a brief explanation of the condition causing the error.
WARNINGS	This section lists warnings about special conditions which could seriously affect your working conditions. This is not a list of diagnostics.
NOTES	This section lists additional information that does not belong anywhere else on the page. It takes the form of an aside to the user, covering points of special interest. Critical information is never covered here.
BUGS	This section describes known bugs and, wherever possible, suggests workarounds.

CHAPTER

File Formats

NAME

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

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, but it is not required to assign values to all eleven parameters. If a value is not assigned, pkgadd(1M) 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 (except the mail parameter) 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 (see NOTES).

basedir
: Indicates the base directory where relocatable packages are to be installed. If there is no basedir entry in the file, the installer will be prompted for a path name, as if the file contained the entry basedir=ask. This parameter can also be set to default (entry is basedir=default). In this instance, the package is installed into the base directory specified by the BASEDIR parameter in the pkginfo(4) file.

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

EXAMPLE 1 Sample of `admin` file.

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), `pkginfo`(4)

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/mail/aliases	Aliases for the local host, in ASCII format. Root can edit this file to add, update, or delete local mail aliases. Additionally, `sendmail`(1M) will build the DBM files for `/etc/mail/aliases` if they are missing, so long as the `/etc/mail/aliases*` files are owned by root *and* root has exclusive write permission.
/etc/mail/aliases. {dir , pag}	The aliasing information from `/etc/mail/aliases` , in binary, `dbm` format for use by `sendmail`(1M) . The program `newaliases`(1) , which is invoked automatically by `sendmail`(1M) , maintains these files. Also, `sendmail`(1M) will build the DBM files for `/etc/mail/aliases` . `{dir, pag}` if they are missing, so long as `/etc/mail/aliases` . `{dir, pag}` is owned by root *and* root has exclusive write permission.
~/.forward	Addresses to which a user's mail is forwarded (see `Automatic Forwarding`).

In addition, the NIS name services aliases map *mail.aliases* , and the NIS+ *mail_aliases* table, both contain 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.

Internet-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.

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/mail/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*

`sendmail` directs error-messages resulting from mail to *aliasname* to *address* , instead of back to the person who sent the message. `sendmail` rewrites the SMTP envelope sender to match this, so `owner-aliasname` should always point to `alias-request` , and `alias-request` should point to the owner's actual address:

```
owner-aliasname:      aliasname-request
aliasname-request     address
```

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.

NIS and NIS+ Domain Aliases

The aliases file on the master NIS server is used for the *mail.aliases* NIS map, which can be made available to every NIS client. The *mail_aliases* table serves the same purpose on a NIS+ server. 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 NIS or NIS+ client could just mail to `jsmith` and not have to remember the machine and username for John Smith.

If a NIS or NIS+ alias does not resolve to an address with a specific host, then the name of the NIS or NIS+ 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 NIS or NIS+ client to `root@podunk-u` if the name of the NIS or NIS+ 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 NIS aliases. Otherwise, copies of the message may "bounce." Usually, the solution is to change the NIS 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`	password file
`/etc/nsswitch.conf`	name service switch configuration file
`/etc/mail/aliases`	mail aliases file (ascii)
`/etc/mail/aliases.dir`	database of mail aliases (binary)
`/etc/mail/aliases.pag`	database of mail aliases (binary)
`/etc/mail/sendmail.cf`	sendmail configuration file
`~/.forward`	forwarding information file

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWsndmr

SEE ALSO

`newaliases`(1) , `passwd`(1) , `uucp`(1C) , `vacation`(1) , `sendmail`(1M) , `dbm`(3UCB) , `getusershell`(3C) , `passwd`(4) , `shells`(4) , `attributes`(5)

NOTES

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

For aliases which result in piping to a program or concatenating a file, the shell of the controlling user must be allowed. Which shells are and are not allowed are determined by `getusershell`(3C) .

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`(3ELF) describes. An overview of the file format follows. For more complete information, see the references given below.

<table>
<tr><th>Linking View</th><th>Execution View</th></tr>
<tr><td>ELF header</td><td>ELF header</td></tr>
<tr><td>Program header table
optional</td><td>Program header table</td></tr>
<tr><td>Section 1</td><td rowspan="2">Segment 1</td></tr>
<tr><td>. . .</td></tr>
<tr><td>Section n</td><td rowspan="2">Segment 2</td></tr>
<tr><td>. . .</td></tr>
<tr><td>. . .</td><td>. . .</td></tr>
<tr><td>Section header table</td><td>Section header table
optional</td></tr>
</table>

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`(3ELF)

ANSI C Programmer's Guide

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_t  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 VMAGLEN 8
#define VVOLLEN 6
#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

EXAMPLE 1 Sample `asetenv` file showing the settings of the ASET configurable parameters

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

EXAMPLE 1 Examples of Valid Entries for the `tune.low` , `tune.med` , and `tune.high` Files

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

```
/bin 00777   root staffsymlink
/etc 02755   root staffdirectory
/dev/sd*  00640  rootoperatorfile
```

SEE ALSO

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

ASET Administrator Manual

NAME

audit_class – audit class definitions

SYNOPSIS

`/etc/security/audit_class`

DESCRIPTION

`/etc/security/audit_class` is an ASCII system file that stores class definitions. Programs use the `getauclassent`(3BSM) 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

EXAMPLE 1 Sample of an `audit_class` file.

Here is a sample of an `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(3BSM), 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.

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

NAME

audit_control – control information for system audit daemon

SYNOPSIS

`/etc/security/audit_control`

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

EXAMPLE 1 Sample /etc/security/audit_control file for the machine eggplant.

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

NOTES

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

NAME

audit_data – current information on audit daemon

SYNOPSIS

/etc/security/audit_data

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

EXAMPLE 1 A sample audit_data file.

```
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)

NOTES

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

NAME

audit_event – audit event definition and class mapping

SYNOPSIS

`/etc/security/audit_event`

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`(3BSM) 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

EXAMPLE 1 Sample of the `audit_event` file entries.

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`(3BSM), `audit_control`(4)

NOTES

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

NAME

audit.log – audit trail file

SYNOPSIS

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

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               1 byte
seconds of time        4 bytes
milliseconds of time   4 bytes
file name length       2 bytes
file pathname          N bytes + 1 terminating NULL byte
```

The `header` token consists of:

```
token ID               1 byte
record byte count      4 bytes
version #              1 byte    [2]
event type             2 bytes
event modifier         2 bytes
seconds of time        4 bytes/8 bytes (32-bit/64-bit value)
milliseconds of time   4 bytes/8 bytes (32-bit/64-bit value)
```

The expanded header token consists of:

```
toke ID                 1 byte
record byte count       4 bytes
version #               1 byte     [2]
event type              2 bytes
event modifier          2 bytes
address type/length     4 bytes
machine address         4 bytes/16 bytes (IPv4/IPv6 address)
seconds of time         4 bytes/8 bytes  (32/64-bits)
milliseconds of time    4 bytes/8 bytes  (32/64-bits)
```

The trailer token consists of:

```
token ID                1 byte
trailer magic number    2 bytes
record byte count       4 bytes
```

The arbitrary data token is defined:

```
token ID                1 byte
how to print            1 byte
basic unit              1 byte
unit count              1 byte
data items              (depends on basic unit)
```

The in_addr token consists of:

```
token ID                1 byte
internet address        4 bytes
```

The expanded in_addr token consists of:

```
token ID                1 byte
IP address type/length  4 bytes
IP address             16 bytes
```

The ip token consists of:

```
token ID                1 byte
version and ihl         1 byte
type of service         1 byte
length                  2 bytes
id                      2 bytes
offset                  2 bytes
ttl                     1 byte
protocol                1 byte
checksum                2 bytes
source address          4 bytes
```

```
destination address     4 bytes
```

The expanded `ip` token consists of:

```
token ID                1 byte
version and ihl         1 byte
type of service         1 byte
length                  2 bytes
id                      2 bytes
offset                  2 bytes
ttl                     1 byte
protocol                1 byte
checksum                2 bytes
address type/type       4 bytes
source address          4 bytes/16 bytes (IPv4/IPv6 address)
address type/length     4 bytes
destination address     4 bytes/16 bytes (IPv4/IPv6 address)
```

The `iport` token consists of:

```
token ID                1 byte
port IP address         2 bytes
```

The `path` token consists of:

```
token ID                1 byte
path length             2 bytes
path                    N bytes + 1 terminating NULL byte
```

The `process` token consists of:

```
token ID                1 byte
audit ID                4 bytes
effective user ID       4 bytes
effective group ID      4 bytes
real user ID            4 bytes
real group ID           4 bytes
process ID              4 bytes
session ID              4 bytes
terminal ID
  port ID               4 bytes/8 bytes (32-bit/64-bit value)
  machine address       4 bytes
```

The expanded `process` token consists of:

```
token ID                1 byte
audit ID                4 bytes
effective user ID       4 bytes
effective group ID      4 bytes
real user ID            4 bytes
real group ID           4 bytes
```

```
process ID              4 bytes
session ID              4 bytes
terminal ID
  port ID               4 bytes/8 bytes (32-bit/64-bit value)
  address type/length   4 bytes
  machine address      16 bytes
```

The `return` token consists of:

```
token ID                1 byte
error number            1 byte
return value            4 bytes/8 bytes (32-bit/64-bit value)
```

The `subject` token consists of:

```
token ID                1 byte
audit ID                4 bytes
effective user ID       4 bytes
effective group ID      4 bytes
real user ID            4 bytes
real group ID           4 bytes
process ID              4 bytes
session ID              4 bytes
terminal ID
  port ID               4 bytes/8 bytes (32-bit/64-bit value)
  machine address       4 bytes
```

The expanded `subject` token consists of:

```
token ID                1 byte
audit ID                4 bytes
effective user ID       4 bytes
effective group ID      4 bytes
real user ID            4 bytes
real group ID           4 bytes
process ID              4 bytes
session ID              4 bytes
terminal ID
  port ID               4 bytes/8 bytes (32-bit/64-bit value)
  address type/length   4 bytes
  machine address      16 bytes
```

The `System V IPC` token consists of:

```
token ID                1 byte
object ID type          1 byte
object ID               4 bytes
```

The `text` token consists of:

```
token ID                1 byte
text length             2 bytes
text                    N bytes + 1 terminating NULL byte
```

The `attribute` token consists of:

```
token ID                1 byte
file access mode        4 bytes
owner user ID           4 bytes
owner group ID          4 bytes
file system ID          4 bytes
node ID                 8 bytes
device                  4 bytes/8 bytes (32-bit/64-bit)
```

The `groups` token consists of:

```
token ID                1 byte
number groups           2 bytes
group list              N * 4 bytes
```

The `System V IPC permission` token consists of:

```
token ID                1 byte
owner user ID           4 bytes
owner group ID          4 bytes
creator user ID         4 bytes
creator group ID        4 bytes
access mode             4 bytes
slot sequence #         4 bytes
key                     4 bytes
```

The `arg` token consists of:

```
token ID                1 byte
argument #              1 byte
argument value          4 bytes/8 bytes (32-bit/64-bit value)
text length             2 bytes
text                    N bytes + 1 terminating NULL byte
```

The `exec_args` token consists of:

```
token ID                1 byte
count                   4 bytes
text                    count null-terminated string(s)
```

The `exec_env` token consists of:

```
token ID                1 byte
count                   4 bytes
text                    count null-terminated string(s)
```

The `exit` token consists of:

```
token ID               1 byte
status                 4 bytes
return value           4 bytes
```

The `socket` token consists of:

```
token ID               1 byte
socket type            2 bytes
remote port            2 bytes
remote Internet address 4 bytes
```

The expanded `socket` token consists of:

```
token ID               1 byte
socket type            2 bytes
local port             2 bytes
address type/length    4 bytes
local Internet address  4 bytes/16 bytes (IPv4/IPv6 address)
remote port            4 bytes
address type/length    4 bytes
remote Internet address 4 bytes/16 bytes (IPv4/IPv6 address)
```

The `seq` token consists of:

```
token ID               1 byte
sequence number        4 bytes
```

SEE ALSO

audit(1M), auditd(1M), bsmconv(1M), audit(2), auditon(2), au_to(3BSM), audit_control(4)

NOTES

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

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

NAME

audit_user – per-user auditing data file

SYNOPSIS

/etc/security/audit_user

DESCRIPTION

audit_user is an access-restricted database that stores per-user auditing preselection data. The audit_user file can be used with other authorization sources, including the NIS map audit_user.byname and the NIS+ table audit_user. Programs use the getauusernam(3BSM) routines to access this information.

The search order for multiple user audit information sources is specified in the /etc/nsswitch.conf file, as described in the nsswitch.conf(4) man page. The lookup follows the search order for passwd(4).

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.

For a complete description of the audit flags and how to combine them, see the audit_control(4) man page.

EXAMPLES

EXAMPLE 1 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/nsswitch.conf
/etc/passwd
/etc/security/audit_user

SEE ALSO

bsmconv(1M), getauusernam(3BSM), audit_control(4), nsswitch.conf(4), passwd(4)

NOTES

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

NAME

auth_attr – authorization description database

SYNOPSIS

/etc/security/auth_attr

DESCRIPTION

/etc/security/auth_attr is a local source for authorization names and descriptions. The auth_attr file can be used with other authorization sources, including the auth_attr NIS map and NIS+ table. Programs use the getauthattr(3SECDB) routines to access this information.

The search order for multiple authorization sources is specified in the /etc/nsswitch.conf file, as described in the nsswitch.conf(4) man page.

An authorization is a right assigned to users that is checked by certain privileged programs to determine whether users can execute restricted functionality. Each entry in the auth_attr database consists of one line of text containing six fields separated by colons (:). Line continuations using the backslash (\) character are permitted. The format of each entry is:

name:*res1*:*res2*:*short_desc*:*long_desc*:*attr*

name The name of the authorization. Authorization names are unique strings. Construct authorization names using the following convention:

prefix. or *prefix.suffix*

prefix Everything in the name field up to the final dot (.). Authorizations from Sun Microsystems, Inc. use solaris as a prefix. To avoid name conflicts, all other authorizations should use a prefix that begins with the reverse–order Internet domain name of the organization that creates the authorization (for example, com.xyzcompany). Prefixes can have additional arbitrary components chosen by the authorization's developer, with components separated by dots.

suffix The final component in the name field. Specifies what is being authorized.

When there is no suffix, the name is defined as a heading. Headings are not assigned to users but are constructed for use by applications in their GUIs.

When a name ends with the word grant, the entry defines a grant authorization. Grant authorizations are used to support fine-grained delegation. Users with appropriate grant authorizations can delegate some of their authorizations to others. To assign an authorization, the user needs to have both the authorization itself and the appropriate grant authorization.

res1 Reserved for future use.

res2 Reserved for future use.

short_desc A short description or terse name for the authorization. This name should be suitable for displaying in user interfaces, such as in a scrolling list in a GUI.

long_desc A long description. This field can explain the precise purpose of the authorization, the applications in which it is used, and the type of user that would be interested in using it. The long description can be displayed in the help text of an application.

attr An optional list of semicolon-separated (`;`) key-value pairs that describe the attributes of an authorization. Zero or more keys may be specified. The keyword `help` identifies a help file in HTML. Help files can be read by a web browser using the URL:

```
file:/usr/lib/help/auths/locale/C/index.html
```

EXAMPLES

EXAMPLE 1 Constructing a name

In the following example, the name has a prefix (`solaris.`) followed by a suffix (`printer`):

```
solaris.printer
```

EXAMPLE 2 Defining a heading

Because the name field ends with a dot, the following entry defines a heading:

```
solaris.hostmgr.:::Computers & Networks::help=HostMgrHeader.html
```

EXAMPLE 3 Assigning separate authorizations to set user attributes

In this example, a heading entry is followed by other associated authorization entries. The entries below the heading provide separate authorizations for setting user attributes. The *attr* field for each entry, including the heading entry, assigns a help file. The application that uses the `help` key requires the value to equal the name of a file ending in `.htm` or `.html`:

```
solaris.usermgr.:::Users, Groups & Email Aliases::help=UserMgrHeader.html
solaris.usermgr.pswd:::Change User Passwords::help=UserMgrPswd.html
solaris.usermgr.write:::Add, Modify & Delete::help=UserMgrWrite.html
```

EXAMPLE 4 Assigning a grant authorization

This example assigns to an administrator the following authorizations:

```
solaris.printmgr.grant
solaris.printmgr.admin
solaris.printmgr.nobanner
solaris.login.enable
```

With the above authorizations, the administrator can assign to others the `solaris.printmgr.admin` and `solaris.printmgr.nobanner` authorizations, but not the `solaris.login.enable` authorization. If the administrator has both the grant authorization, `solaris.printmgr.grant`, and the wildcard authorization, `solaris.printmgr.*`, the administrator can grant to others any of the printer authorizations. See `user_attr`(4) for more information about how wildcards can be used to assign multiple authorizations whose names begin with the same components.

EXAMPLE 5 Authorizing the ability to assign other authorizations

The following entry defines an authorization that grants the ability to assign any authorization created with a `solaris` prefix, when the administrator also has either the specific authorization being granted or a matching wildcard entry:

```
solaris.grant:::Grant All Rights::help=PriAdmin.html
```

EXAMPLE 6 Consulting the local authorization file ahead of the NIS table

With the following entry from `/etc/nsswitch.conf`, the local `auth_attr` file is consulted before the NIS table:

```
auth_attr:files nisplus
```

FILES

`/etc/nsswitch.conf`

`/etc/user_attr`

`/etc/security/auth_attr`

NOTES

When deciding which authorization source to use (see DESCRIPTION), keep in mind that NIS+ provides stronger authentication than NIS.

Because the list of legal keys is likely to expand, any code that parses this database must be written to ignore unknown key-value pairs without error. When any new keywords are created, the names should be prefixed with a unique string, such as the company's stock symbol, to avoid potential naming conflicts.

Each application has its own requirements for whether the help value must be a relative pathname ending with a filename or the name of a file. The only known requirement is for the name of a file.

The following characters are used in describing the database format and must be escaped with a backslash if used as data: colon (`:`), semicolon (`;`), equals (=), and backslash (\).

SEE ALSO

`getauthattr`(3SECDB), `getexecattr`(3SECDB), `getprofattr`(3SECDB), `getuserattr`(3SECDB), `exec_attr`(4), `nsswitch.conf`(4), `user_attr`(4)

NAME

bootparams – boot parameter data base

SYNOPSIS

`/etc/bootparams`

DESCRIPTION

The `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 `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-specifier* . . .

The first item of each entry is the host name of the diskless client. 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.

This is followed by one or more whitespace characters and a series of identifier-specifiers separated by whitespace characters.

Each identifier-specifier has the form:

identifier=*server*:*pathname*

or

identifier=*domain-name*

The first form is used for file-specific identifiers. A file-specific *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-specifier.

Non-file-specific identifiers use the second form of identifier-specifier. One non-file-specific value for *identifier* is supported: the assignment of the client's domain name. In this case, the value used for *identifier* is `domain`. *domain-name* must be the client's domain name. The algorithm for determining a client's domain name is to first check for a `domain` identifier in the client-specific entry and then in "wildcard" entry. If none is found, the server's domain name is used.

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.

A variation of the first form (*identifier=server*:*pathname*) is used for the `ns` key which forces `sysidtool`(1M) to use a specific name service. By default, `sysidtool` uses NIS+ in preference to NIS if it can find a NIS+ server for the system's domain on the subnet. This key may be necessary if you are trying to set up a hands-off installation, or if the name server is on a different subnet, which is common with NIS+.

If this key is not used, `sysidtool` uses broadcast to attempt to bind to either a NIS+ or NIS server; if a name server is not on the local subnet, which is possible for NIS+, the bind will fail, automatic configuration of the name service will fail, and an interactive screen is displayed, prompting the user to specify the name service.

The `ns` entry has the form:

```
ns=[server] : [nameservice] [(netmask)]
```

where:

server — the name of a server that will provide a name service to bind to

nameservice — the name service (`nis`, `nisplus`, or `none`);

netmask — a series of four numbers separated by periods that specifies which portion of an IP address is the network part, and which is the host part.

The `ns` keyword can be set in `add_install_client` or by Host Manager.

EXAMPLES

EXAMPLE 1 Example Of An Entry In The `bootparams` File

Here is an example of an entry in the `bootparams` file:

```
client1 root=server1:/export/client1/root \
  swap=server1:/export/client1/swap \
  domain=bldg1.workco.com
  root=server2:/export/client2/root ns=:nis
  root=server2:/export/client2/root ns=watson:
                root=server2:/export/client2/root
ns=mach:nisplus(255.255.255.0)
```

FILES

`/etc/bootparams`

SEE ALSO

`rpc.bootparamd`(1M), `sysidtool`(1M), `nsswitch.conf`(4)

IA 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 IA 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_2.6/Product`. 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`(1M) concatenates the two values (inserting a space) to produce the name displayed in its software selection menu, for example, `Solaris 2.6`. For unbundled products the combined length of the values of *PRODNAME* and *PRODVERS* must not exceed `256` ASCII characters.

When you install OS services with Solstice Host Manager, directories for diskless clients and Autoclient systems are created by constructing names derived from a concatenation of the values of *PRODNAME*, *PRODVERS*, and client architecture, for example, `/export/exec/Solaris_2.`*x*`_sparc.all/usr/platform`. 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 2.4), the values of *PRODNAME* and *PRODVERS* are effectively restricted to lengths much less than `256`.

The initial install programs and `swmtool`(1M) use the value of the *PRODDIR* macro in the `.cdtoc` file to indicate where packages can be found.

EXAMPLES

EXAMPLE 1 Sample of `.cdtoc` file.

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 2.6 includes the following `.cdtoc` file:

```
PRODNAME=Solaris
PRODVERS=2.6
PRODDIR=Solaris_2.6/Product
```

This file corresponds to the following directory layout on slice 0 of the Solaris 2.6 product CD:

```
/.cdtoc
/Solaris_2.6/Product
     ./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 granularity. 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.

`SUNW_CSRMBRIFF`
Indicates that the package is to be included dynamically in the (meta-)cluster currently being described. The value of this parameter must follow the following format:

```
SUNW_CSRMBRIFF=(<test> <test_arc>)<package>
```

This line will be converted into a *SUNW_CSRMEMBER* entry at media installation time if the test provided matches the platform on which the media is being installed. There may be zero or more *SUN_CSRMBRIFF* parameters per (meta-)cluster.

`SUNW_CSRMBRIFF=`*(<test> <value>)<package>*
where the the <test> is either the builtin test of "platform" or a shell script which returns shell true (0) or shell false (1) depending on the tests being performed in the script. <value> is passed to the test as the first argument and can be used to create a script that tests for multiple hardware objects. Finally <package> is the package that will be included in the final `.clustertoc` file as a *SUNW_CSRMEMBER*. See `parse_dynamic_clustertoc`(1M) for more information about the scripts.

EXAMPLES

EXAMPLE 1 A Cluster Description

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

```
CLUSTER=SUNWCacc
NAME=System Accounting
DESC=System accounting utilities
VENDOR=Sun Microsystems, Inc.
VERSION=7.2
SUNW_CSRMEMBER=SUNWaccr
SUNW_CSRMEMBER=SUNWaccu
END
```

EXAMPLE 2 A Meta-cluster Description

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

```
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.x
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
```

EXAMPLE 3 A Meta-cluster Description With a Dynamic Cluster Entry

The following is an example of a meta-cluster description with a dynamic cluster entry as indicated by the use of the SUNW_CSRMBRIFF parameter entries.

```
METACLUSTER=SUNWCprog
NAME=Developer System Support
DESC=A pre-defined software configuration consisting of the
typical software used by software developers.
VENDOR=Sun Microsystems, Inc.
VERSION=2.5
SUNW_CSRMEMBER=SUNWCadm
SUNW_CSRMBRIFF=(smcc.dctoc tcx)SUNWCtcx
SUNW_CSRMBRIFF=(smcc.dctoc leo)SUNWCleo
SUNW_CSRMBRIFF=(smcc.dctoc sx)SUNWCsx
 . . .
END
```

SEE ALSO

`parse_dynamic_clustertoc`(1M), `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

EXAMPLE 1 Sample `compver` file.

A sample `compver` file is shown below:

```
Version 1.3
Version 1.0
```

SEE ALSO

`pkginfo`(4)

Application Packaging Developer's Guide

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.

The entries in the `compver` file must match the values assigned to the `VERSION` parameter in the `pkginfo`(4) files.

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) are displayed on the terminal at the time of package installation.

SEE ALSO

Application Packaging Developer's Guide

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. This is also true for a process with an effective group ID different from the real group ID. Set-user-ID and set-group-ID programs do not produce core images either when they terminate, since this would cause a security loophole.

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 segment that was part of the process address space, including shared library segments. The contents of the writable segments are also part of the core image.

The program header of an ELF core file also contains entries for two `NOTE` segments, each containing several note entries as described below. The note entry header and core file note type (`n_type`) definitions are contained in `<sys/elf.h>`. The first `NOTE` segment exists for binary compatibility with old programs that deal with core files. It contains structures defined in `<sys/old_procfs.h>`. New programs should recognize and skip this `NOTE` segment, advancing instead to the new `NOTE` segment. The old `NOTE` segment will be deleted from core files in a future release.

The old `NOTE` segment contains the following entries. Each has entry name `"CORE"` and presents the contents of a system structure:

`prpsinfo_t`
: `n_type` : `NT_PRPSINFO`. This entry 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/old_procfs.h>`.

`char` array
: `n_type`: `NT_PLATFORM`. This entry contains a string describing the specific model of the hardware platform on which this core file was created. This information is the same as provided by `sysinfo`(2) when invoked with the command `SI_PLATFORM`.

auxv_t array	n_type: NT_AUXV. This entry contains the array of auxv_t structures that was passed by the operating system as startup information to the dynamic linker. Auxiliary vector information is defined in <sys/auxv.h>.

Following these entries, for each *light-weight process* (LWP) in the process, the old NOTE segment contains an entry with a prstatus_t structure, plus other optionally-present entries describing the LWP, as follows:

prstatus_t	n_type: NT_PRSTATUS. This structure contains things of interest to a debugger from the operating system, such as the general registers, signal dispositions, state, reason for stopping, process-ID, and so forth. The prstatus_t structure is defined in <sys/old_procfs.h>.
prfpregset_t	n_type: NT_PRFPREG. This entry is present only if the LWP used the floating-point hardware. It contains the floating-point registers. The prfpregset_t structure is defined in <sys/procfs_isa.h>.
gwindows_t	n_type: NT_GWINDOWS. This entry is present only on a SPARC machine and only if the system was unable to flush all of the register windows to the stack. It contains all of the unspilled register windows. The gwindows_t structure is defined in <sys/regset.h>.
prxregset_t	n_type: NT_PRXREG. This entry is present only if the machine has extra register state associated with it. It contains the extra register state. The prxregset_t structure is defined in <sys/procfs_isa.h>.

The new NOTE segment contains the following entries. Each has entry name “CORE” and presents the contents of a system structure:

psinfo_t	n_type: NT_PSINFO. This structure 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 psinfo_t structure is defined in <sys/procfs.h>.

pstatus_t	n_type: NT_PSTATUS. This structure contains things of interest to a debugger from the operating system, such as pending signals, state, process-ID, and so forth. The pstatus_t structure is defined in <sys/procfs.h>.
char array	n_type: NT_PLATFORM. This entry contains a string describing the specific model of the hardware platform on which this core file was created. This information is the same as provided by sysinfo(2) when invoked with the command SI_PLATFORM.
auxv_t array	n_type: NT_AUXV. This entry contains the array of auxv_t structures that was passed by the operating system as startup information to the dynamic linker. Auxiliary vector information is defined in <sys/auxv.h>.
struct utsname	n_type: NT_UTSNAME. This structure contains the system information that would have been returned to the process if it had performed a uname(2) system call prior to dumping core. The utsname structure is defined in <sys/utsname.h>.
prcred_t	n_type: NT_PRCRED. This structure contains the process credentials, including the real, saved, and effective user and group IDs. The prcred_t structure is defined in <sys/procfs.h>. Following the structure is an optional array of supplementary group IDs. The total number of supplementary group IDs is given by the pr_ngroups member of the prcred_t structure, and the structure includes space for one supplementary group. If pr_ngroups is greater than 1, there will be pr_ngroups - 1 gid_t items following the structure; otherwise, there will be no additional data.

Following these entries, for each LWP in the process, the new NOTE segment contains an entry with an lwpsinfo_t structure plus an entry with an lwpstatus_t structure, plus other optionally-present entries describing the LWP, as follows:

lwpsinfo_t	n_type: NT_LWPSINFO. This structure contains information of interest to the ps(1) command, such as LWP status, CPU

usage, "nice" value, LWP-ID, and so forth. The `lwpsinfo_t` structure is defined in `<sys/procfs.h>`.

`lwpstatus_t`	`n_type`: `NT_LWPSTATUS`. This structure contains things of interest to a debugger from the operating system, such as the general registers, the floating point registers, state, reason for stopping, LWP-ID, and so forth. The `lwpstatus_t` structure is defined in `<sys/procfs.h>`.
`gwindows_t`	`n_type`: `NT_GWINDOWS`. This entry is present only on a SPARC machine and only if the system was unable to flush all of the register windows to the stack. It contains all of the unspilled register windows. The `gwindows_t` structure is defined in `<sys/regset.h>`.
`prxregset_t`	`n_type`: `NT_PRXREG`. This entry is present only if the machine has extra register state associated with it. It contains the extra register state. The `prxregset_t` structure is defined in `<sys/procfs_isa.h>`.
`asrset_t`	`n_type`: `NT_ASRS`. This entry is present only on a SPARC V9 machine and only if the process is a 64-bit process. It contains the ancillary state registers for the LWP. The `asrset_t` structure is defined in `<sys/regset.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), `ps`(1), `crash`(1M), `getrlimit`(2), `setuid`(2), `sysinfo`(2), `uname`(2), `elf`(3ELF), `a.out`(4), `proc`(4), `signal`(3HEAD)

ANSI C Programmer's Guide

NAME

dacf.conf – device auto-configuration configuration file

SYNOPSIS

`/etc/dacf.conf`

DESCRIPTION

The kernel uses the `dacf.conf` file to automatically configure hot plugged devices. Because the `dacf.conf` file contains important kernel state information, it should not be modified.

The format of the `/etc/dacf.conf` file is not public and might change in versions of the Solaris operating environment that are not compatible with Solaris 8.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsr

SEE ALSO

`attributes`(5)

NOTES

This document does not constitute an API. The `/etc/dacf.conf` file might not exist or might contain different contents or interpretations in versions of the Solaris operating environment that are not compatiable with Solaris 8. The existence of this notice does not imply that any other documentation lacking this notice constitutes an API.

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

`fsck`(1M) , `fstypes`(4) , `vfstab`(4)

NAME

defaultrouter – configuration file for default router(s)

SYNOPSIS

/etc/defaultrouter

DESCRIPTION

The /etc/defaultrouter file defines the default routers the system will use.

The format of the file is as follows:

The /etc/defaultrouter file can contain the hostnames or IP addresses of one or more default routers, separated by white space. If you use hostnames, each hostname must also be listed in the local /etc/hosts file, because no name services are running at the time that this script is run.

Lines beginning with the "#" character are treated as comments.

The default routes listed in this file replace those added by the kernel during diskless booting. An empty /etc/defaultrouter file will cause the default route added by the kernel to be deleted.

FILES

/etc/defaultrouter	Configuration file containing the hostnames or IP addresses of one or more default routers.

SEE ALSO

hosts(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 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 each *(arch)version* pair must begin on a new line

that begins with white space. A null version set equates to any version of the indicated package.

EXAMPLES

EXAMPLE 1 Sample of depend file.

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
```

SEE ALSO

Application Packaging Developer's Guide

NAME

device_allocate – device_allocate file

SYNOPSIS

`/etc/security/device_allocate`

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;*auths*;*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.

auths This field contains a comma-separated list of authorizations required to allocate the device, or asterisk (`*`) to indicate that the device is *not* allocatable, or an '`@`' symbol to indicate that no explicit authorization is needed to allocate the device.

The default authorization is `solaris.device.allocate`. See `auths`(1)

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 '\'.

White space is allowed in any field.

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

EXAMPLE 1 Declaring an allocatable device

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;\
      solaris.device.allocate;\
      /etc/security/lib/st_clean;\
```

EXAMPLE 2 Declaring an allocatable device with authorizations

Declare that physical device fd0 is of type fd. fd is allocatable by users with the solaris.device.allocate authorization, 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;\
      &;\
      /etc/security/lib/fd_clean;\
```

Notice that making a device allocatable means that you need to allocate and deallocate it to use it (with allocate(1M) and deallocate(1M)). If a device is not allocatable, there will be an asterisk (*) in the *auths* field, and no one can use the device.

FILES

/etc/security/device_allocate Contains list of allocatable devices

SEE ALSO

auths(1), allocate(1M), bsmconv(1M), deallocate(1M), list_devices(1M), auth_attr(4)

NOTES

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

NAME

device.cfinfo – devconfig configuration files

SYNOPSIS

`device.cfinfo`

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

EXAMPLE 1 Device configuration file `logi.cfinfo` for the LOGITECH bus mouse.

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"
 __owname__="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 Desktop 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`

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	IA

SEE ALSO

`devconfig`(1M), `driver.conf`(4), `attributes`(5) *OpenWindows Desktop Reference Manual*

NAME

device_maps – device_maps file

SYNOPSIS

`/etc/security/device_maps`

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 bef\ore device allocation is enabled.

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

EXAMPLES

EXAMPLE 1 A sample `device_maps` file

```
# 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)

NOTES

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

NAME

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

dhcp – file containing default parameter values for the DHCP service

DESCRIPTION

The dhcp file resides in directory /etc/default and contains parameters for specifying the type and location of DHCP service databases, as well as DHCP service daemon default settings.

The dhcp file format is ASCII; comment lines begin with the crosshatch (#) character. Parameters consist of a keyword followed by an equals (=) sign followed by the parameter value, of the form:

Keyword=*Value*

Two parameters are currently supported:

Keyword	*Value*
RESOURCE	Can be either nisplus or files
PATH	Path to data files. The value of the PATH keyword is specified as an absolute path for the files resource, or a fully-qualified directory for the nisplus resource.
RUN_MODE	server or relay. Selects daemon run mode. Default is server.
VERBOSE	TRUE / FALSE. Toggles verbose mode. Default is FALSE. Generic parameter.
RELAY_HOPS	Integer. Max number of BOOTP relay hops before packet is dropped. Default is 4. Generic parameter.
INTERFACES	String. Comma-separated list of interface names to listen to. Generic parameter.
LOGGING_FACILITY	Integer. Local facility number (0–7 includive) to log DHCP events to. Default is not to log transactions. Generic parameter.
ETHERS_COMPAT	TRUE / FALSE. Toggles ethers compatibility mode. Default is TRUE. server mode only parameter.
ICMP_VERIFY	TRUE / FALSE. Toggles ICMP echo verification of IP addresses. Default is TRUE. server mode only parameter.
OFFER_CACHE_TIMEOUT	Integer. Number of seconds before OFFER cache timeoutsoccur. Default is 10 seconds. server mode only parameter.

Keyword	*Value*
RESCAN_INTERVAL	Integer. Number of minutes between automatic dhcptab rescans. Default is not to do rescans. server mode only parameter.
BOOTP_COMPAT	String automatic or manual. Enable BOOTP compatibility. Default is no BOOTP. Value selects BOOTP address allocation method. automatic for dynamic allocation, manual for static allocation. server mode only parameter.
RELAY_DESTINATIONS	String. Comma-separated list of hostnames and IP addresses of relay destinations. relay mode only parameter.

The preferred method of modifying the dhcp file is through use of the in.dhcpd(1M) or dhcpconfig(1M) utilities.

SEE ALSO

dhcpconfig(1M), dhcpmgr(1M), in.dhcpd(1M)

NAME dhcp_inittab – information repository for DHCP options

SYNOPSIS `/etc/dhcp/inittab`

DESCRIPTION DHCP options are network configuration parameters passed from DHCP servers to DHCP clients when a client machine uses DHCP. Since many DHCP-related commands must parse and understand these DHCP options, this file serves as a central location where information about these options may be obtained.

The `dhcp_inittab` file provides three general pieces of information:

- It provides a mnemonic alias for each option number. For instance, option 12 is aliased to the name "Hostname". This is useful for DHCP-related programs which require human interaction, such as `dhcpinfo`(1).
- It provides information about the syntax for each option. This includes information such as the type of the value, for example, whether it is a 16-bit integer or an IP address.
- It provides the policy for what options are visible to which DHCP-related programs.

Each DHCP option belongs to a certain category, which roughly defines the scope of the option; for instance, an option may only be understood by certain hosts within a given site, or it may be globally understood by all DHCP clients and servers. The following categories are defined; the category names are not case-sensitive:

`STANDARD` All client and server DHCP implementations agree on the semantics. These are administered by the Internet Assigned Numbers Authority (“IANA”). These options are numbered from 1 to 127.

`SITE` Within a specific site, all client and server implementations agree on the semantics. However, at another site the type and meaning of the option may be quite different. These options are numbered from 128 to 254.

`VENDOR` Each vendor may define 254 options unique to that vendor. The vendor is identified within a DHCP packet by the "Vendor Class" option, number 60. An option with a specific numeric identifier belonging to one vendor will, in general, have a type and semantics different from that of a different vendor. Vendor options are "super-encapsulated" into the vendor field number 43, as defined in *RFC 2132*.

`FIELD` This category allows the fixed fields within a DHCP packet to be aliased to a mnemonic name for use with `dhcpinfo`(1).

`INTERNAL`	This category is internal to the Solaris DHCP implementation and will not be further defined.

USAGE

Data entries are written one per line and have seven fields; each entry provides information for one option. Each field is separated by a comma, except for the first and second, which are separated by whitespace (as defined in `isspace`(3C)). An entry cannot be continued onto another line. Blank lines and those whose first non-whitespace character is '#' are ignored.

The fields, in order, are:

- Mnemonic Identifier

  The Mnemonic Identifier is a user-friendly alias for the option number; it is not case sensitive. This field must be per-category unique and should be unique across all categories. The option names in the `STANDARD`, `SITE`, and `VENDOR` spaces should not overlap, or the behavior will be undefined.

- Category (scope)

  The Category field is one of `STANDARD`, `SITE`, `VENDOR`, `FIELD`, or `INTERNAL` and identifies the scope in which the option falls.

- Option Number

  The Option Number is the number of this option when it is in a DHCP packet. This field should be per-category unique and the `STANDARD` and `SITE` fields should not have overlapping code fields or the behavior is undefined.

- Data Type

  Data Type is one of the follow values, which is not case sensitive:

`Ascii`	A printable character string
`Octet`	An array of bytes
`Unumber8`	An 8-bit unsigned integer
`Snumber8`	An 8-bit signed integer
`Unumber16`	A 16-bit unsigned integer
`Snumber16`	A 16-bit signed integer
`Unumber32`	A 32-bit unsigned integer
`Snumber32`	A 32-bit signed integer
`Unumber64`	A 64-bit unsigned integer
`Snumber64`	A 64-bit signed integer

`Ip` An IP address

The data type field describes an indivisible unit of the option payload, using one of the values listed above.

- Granularity

  The Granularity field describes how many "indivisible units" in the option payload make up a whole value or item for this option.

- Maximum Number Of Items
- Visibility

  The Visibility field specifies which DHCP-related programs make use of this information, and should always be defined as "`sdmi`" for newly added options.

EXAMPLES

EXAMPLE 1 Altering the `dhcp_inittab` File

In general, the `dhcp_inittab` file should only be altered to add either a DHCP `STANDARD` option or `SITE` option. For instance:

```
ipPairs    SITE, 132, IP, 2, 0, sdmi
```

describes an option named `ipPairs`, that is in the `SITE` category. That is, it is defined by each individual site, and is option code 132, which is of type IP Address, consisting of a potentially infinite number of pairs of IP addresses.

FILES

`/etc/dhcp/inittab`

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsr
Interface Stability	Unstable

SEE ALSO

`dhcpinfo`(1), `dhcpagent`(1M), `attributes`(5)

Alexander, S., and R. Droms, *RFC 2132, DHCP Options and BOOTP Vendor Extensions*, Network Working Group, March 1997.

Droms, R., *RFC 2131, Dynamic Host Configuration Protocol*, Network Working Group, March 1997.

NAME

dhcp_network – dhcp network DHCP database

DESCRIPTION

The `dhcp network` database is used to map a Dynamic Host Configuration Protocol (DHCP) client's client identifier to an IP address and the associated configuration parameters of that address. This database is located by the DHCP server at runtime upon receipt of a BOOTP request.

The `dhcp network` databases can exist as NIS+ tables or ASCII files. Since the format of the file could change, the preferred method of managing the `dhcp network` databases is through the use of the `pntadm`(1M) command.

Each entry in a `dhcp network` database has the form:

Client_ID	Flags	Client_IP	Server_IP	Lease	Macro	#Comment

The fields are defined as follows:

`Client_ID` The client identifier field, `Client_ID`, is an ASCII hexadecimal representation of the unique octet string value of the DHCP Client Identifier Option (code 61) which identifies a DHCP client. In the absence of the DHCP Client Identifier Option, the DHCP client is identified using the form given below for BOOTP clients. The number of characters in this field must be an even number, with a maximum length of 64 characters. Valid characters are `0 - 9` and `A-F`. Entries with values of `00` are freely available for dynamic allocation to requesting clients. BOOTP clients are identified by the concatenation of the network's hardware type (as defined by RFC 1340, titled "Assigned Numbers") and the client's hardware address. For example, the following BOOTP client has a hardware type of '`01`' (10mb ethernet) and a hardware address of `8:0:20:11:12:b7`, so its client identifier would be: `010800201112B7`

`Flags` The `Flags` field is a decimal value, the bit fields of which can have a combination of the following values:

`1 (PERMANENT)`

Evaluation of the `Lease` field is turned off (lease is permanent). If this bit is not set, Evaluation of the `Lease` field is enabled and the `Lease` is `DYNAMIC.`

`2 (MANUAL)`

This entry has a manual client ID binding (cannot be reclaimed by DHCP server). Client will not be allocated another address.

4 (UNUSABLE)

When set, this value means that either through ICMP echo or client DECLINE, this address has been found to be unusable. Can also be used by the network administrator to *prevent* a certain client from booting, if used in conjunction with the MANUAL flag.

8 (BOOTP)

This entry is reserved for allocation to BOOTP clients only.

Client_IP — The Client_IP field holds the IP address for this entry. This value must be unique in the database.

Server_IP — This field holds the IP address of the DHCP server which *owns* this client IP address, and thus is responsible for initial allocation to a requesting client.

Lease — This numeric field holds the entry's absolute lease expiration time, and is in seconds since January 1, 1970. It can be decimal, or hexadecimal (if 0x prefixes number). The special value -1 is used to denote a permanent lease.

Macro — This ASCII text field contains the dhcptab macro name used to look up this entry's configuration parameters in the dhcptab(4) database.

Comment — This ASCII text field contains an optional comment.

TREATISE ON LEASES

This section describes how the DHCP/BOOTP server calculates a client's configuration lease using information contained in the dhcptab(4) and dhcp network databases. The server consults the LeaseTim and LeaseNeg symbols in the dhcptab, and the Flags and Lease fields of the chosen dhcp network database record.

The server first examines the Flags field for the identified dhcp network record. If the PERMANENT flag is on, then the client's lease is considered permanent.

If the PERMANENT flag is not on, then the server checks if the client's lease as represented by the Lease field in the dhcp network record has expired. If not, then the server checks if the client has requested a new lease. If the LeaseNeg symbol has not been included in the client's dhcptab parameters, then the client's requested lease extension is ignored, and the lease is set to be the time remaining as shown by the Lease field. If the LeaseNeg symbol *has* been included, then the server will extend the client's lease to the value it requested if this requested lease is less than or equal to the current time plus the value of the client's LeaseTim dhcptab parameter.

If the client's requested lease is greater than policy allows (value of `LeaseTim`), then the client is given a lease equal to the current time plus the value of `LeaseTim`. If `LeaseTim` is not set, then the default `LeaseTim` value is one hour.

For more information about the `dhcptab` symbols discussed in this section, see `dhcptab`(4).

EXAMPLES

EXAMPLE 1 Database entry for dynamic allocation.

The following `dhcp network` database entry is free for dynamic allocation. The IP address for this entry is `10.0.0.5`, the IP address of the DHCP server that can initially allocate this address is `10.0.0.1`, the lease expires `754012553`, or `Mon Nov 22 18:55:53 1993`, and the `dhctab` macro associated with this entry is called `10netnis`:

```
00  0   10.0.0.5   10.0.0.1   754012553 10netnis
```

EXAMPLE 2 Manually administered entry with a permanent lease.

The following entry shows a manually administered entry for client ID `010000C0EFA4A`, which has a permanent lease (that is, `MANUAL | PERMANENT == 3`):

```
010000C0EFA4A  3   10.0.0.25   10.0.0.1  -1   10netnis
```

EXAMPLE 3 Manually administered unusable entry.

The following entry shows a `MANUAL` entry which has been marked as `UNUSABLE` (that is, `MANUAL | UNUSABLE == 6`):

```
0408072097C9F  6  10.0.0.26  10.0.0.1   764258362  10netdns
```

EXAMPLE 4 Previously unused DYNAMIC entry.

The following entry for IP address `10.0.0.27` shows a previously unused, `DYNAMIC` entry which uses `dhcptab` macro `10netnis` and is owned by DHCP server `10.0.0.2`:

```
00  0   10.0.0.27  10.0.0.2  0  10netnis
```

EXAMPLE 5 Reserved entry.

The following entry is reserved for BOOTP clients:

```
00  08   10.0.0.27  10.0.0.3  0  10netnis
```

FILES

`/var/dhcp/NNN_NNN_NNN_NNN`
Where `NNN_NNN_NNN_NNN` are database file(s) or NIS+ tables(s).

`/var/dhcp/dhcptab`
file or NIS+ table

SEE ALSO

`dhcpconfig`(1M), `dhcpmgr`(1M), `dhtadm`(1M), `in.dhcpd`(1M), `pntadm`(1M), `dhcptab`(4)

Reynolds, J. and J. Postel, *Assigned Numbers*, STD 2, RFC 1340, USC / Information Sciences Institute, July 1992,

NAME

dhcptab – DHCP configuration parameter table

DESCRIPTION

The `dhcptab` macro table allows network administrators to organize groups of configuration parameters as macro definitions, which can then be further used in the definition of other useful macros. These macros can be configured such that the DHCP server will return their values to DHCP and BOOTP clients.

The preferred method of managing the `dhcptab` macro table is through the use of the `dhtadm`(1M) utility. The syntax described in the balance of this manual page is intended for informational purposes.

Syntax of the dhcptab Table

The syntax of the `dhcptab` table is as follows:

Comments begin with the cross-hatch (#) character in the first position on the line and end with a carriage return. Lines can be continued by escaping the carriage return character with a backslash (\) character.

`dhcptab` records contain three (3) fields:

`Name`	`Type`	`Value`

The fields are defined as follows:

`Name` — This field identifies the record and is used as the search key into the `dhcptab` table. A `Name` must consist of ASCII characters. If the record is of type `Macro`, then the length is limited to 64 characters. If the record is of type `Symbol`, then the length is limited to 8 characters.

`Type` — This field specifies the type of record. Currently, there are only two legal values for `Type`:

- `m (Macro)` — This record is a DHCP macro definition.
- `s (Symbol)` — This record is a DHCP symbol definition. It is used to define vendor and site-specific options.

`Value` — This field contains the value for the specified type of record. For the `macro` type, the value will consist of a series of symbol=value pairs, separated by the colon (`:`) character. For the `symbol` type, the value will consist of a series of fields, separated by a comma (`,`), which define a symbol's characteristics. Once defined, a symbol can be used in macro definitions.

Symbol Characteristics

The fields describing the characteristics of a symbol are as follows:

`Context`	`Code`	`Type`	`Granularity`	`Maximum`

These fields are defined as follows:

`Context` This field defines the context in which the symbol definition is to be used. It can have three values:

`Extend`

This symbol defines a standard option, codes from 77-127. The use of this symbol type is for adding new standard options added since the release of the `dhcp` server.

`Site`

This symbol defines a site-specific option, codes 128-254.

`Vendor=Client Class ...`

This symbol defines a vendor-specific option, codes 1-254. The Vendor context takes ASCII string arguments which identify the client class that this vendor option is associated with. Multiple client class names can be specified, separated by white space. Only those clients whose client class matches one of these values will see this option.

`Code` This field specifies the option code number associated with this symbol. Valid values are 128-254 for site-specific options, and 1-254 for vendor-specific options.

`Type` This field defines the type of data expected as a value for this symbol. Legal values are:

`ASCII` NVT ASCII text. Value is enclosed in double-quotes ("). Granularity setting has no effect on symbols of this type, since ASCII strings have a natural granularity of one (1).

`BOOLEAN` No value is associated with this data type. Presence of symbols of this type denote boolean `TRUE,` whereas absence denotes `FALSE.` Granularity and Miximum values have no meaning for symbols of this type.

`IP` Dotted decimal form of an Internet address. Multi-IP address granularity is supported.

NUMBER — An unsigned number with a supported granularity of 1, 2, 4, and 8 octets.

OCTET — Uninterpreted ASCII representation of binary data. The client identifier is one example of an OCTET string. Valid characters are 0–9, [a-f] [A-F]. One ASCII character represents one nibble (4 bits), thus two ASCII characters are needed to represent an 8 bit quantity. The granularity setting has no effect on symbols of this type, since OCTET strings have a natural granularity of one (1).

Granularity — This value specifies how many objects of Type define a single instance of the symbol value. For example, the static route option is defined to be a variable list of routes. Each route consists of two IP addresses, so the Type is defined to be IP, and the data's granularity is defined to be 2 IP addresses. The granularity field affects the IP and NUMBER data types.

Maximum — This value specifies the maximum items of Granularity which are permissible in a definition using this symbol. For example, there can only be one IP address specified for a subnet mask, so the Maximum number of items in this case is one (1). A Maximum value of zero (0) means that a variable number of items is permitted.

The following example defines a site-specific option called MystatRt, of code 130, type IP, and granularity 2, and a Maximum of 0. This definition corresponds to the internal definition of the static route option (StaticRt).

```
MystatRt s Site,130,IP,2,0
```

Macro Definitions

The following example illustrates a macro defined using the MystatRt site option symbol just defined:

```
10netnis m :MystatRt=3.0.0.0 10.0.0.30:
```

Macro records can be specified in the Macro field in dhcp network databases (see dhcp_network(4)), which will bind particular macro definitions to specific IP addresses.

If present, four macro definitions are consulted by the DHCP server to determine the options that are returned to the requesting client:

`Client Class`	`Network`	`IP Address`	`Client Identifier`

These macros are processed as follows:

`Client Class` A macro called by the ASCII representation of the client class is searched for in the `dhcptab`. If found, then its symbol/value pairs will be selected for delivery to the client. This mechanism permits the network administrator to select configuration parameters to be returned to all clients of the same class.

`Network` A macro named by the dotted Internet form of the network address of the client's network (for example, `10.0.0.0`) is searched for in the `dhcptab`. If found, then its symbol/value pairs will be combined with those of the `Client Class` macro. If a symbol exists in both macros, then the `Network` macro value overrides the value defined in the `Client Class` macro. This mechanism permits the network administrator to select configuration parameters to be returned to all clients on the same network.

`IP Address` This macro is specified in the `dhcp network` database for the record assigned to the requesting client. If this macro is found in the `dhcptab`, then its symbol/value pairs will be combined with those of the `Client Class` macro and the `Network` macro. This mechanism permits the network administrator to select configuration parameters to be returned to clients using a particular IP address. It can also be used to deliver a macro defined to include "server-specific" information by including this macro definition in all `dhcp network` database entries owned by a specific server.

`Client Identifier` A macro named by the ASCII representation of the client's unique identifier as shown in the `dhcp` network table, `dhcp_network`(4). If found, its symbol/value pairs are combined to the sum of the `Client Class`, `Network`, and `IP Address` macros. Any symbol collisions are replaced with those specified in the client identifier macro. This mechanism permits the network administrator to select configuration parameters to be returned to a particular client, regardless of what network that client is connected to.

Internal Symbol Names

The following table maps the available internal symbol names to RFC-2132 options:

Symbol	*Code*	*Description*
`Subnet`	1	Subnet Mask, dotted Internet address (IP).
`UTCoffst`	2	Coordinated Universal time offset (seconds).
`Router`	3	List of Routers, IP.
`Timeserv`	4	List of RFC-868 servers, IP.
`IEN116ns`	5	List of IEN 116 name servers, IP.
`DNSserv`	6	List of DNS name servers, IP.
`Logserv`	7	List of MIT-LCS UDP log servers, IP.
`Cookie`	8	List of RFC-865 cookie servers, IP.
`Lprserv`	9	List of RFC-1179 line printer servers, IP.
`Impress`	10	List of Imagen Impress servers, IP.
`Resource`	11	List of RFC-887 resource location servers, IP.
`Hostname`	12	Client's hostname, value from hosts database.
`Bootsize`	13	Number of 512 octet blocks in boot image, NUMBER.
`Dumpfile`	14	Path where core image should be dumped, ASCII.
`DNSdmain`	15	DNS domain name, ASCII.

Symbol	*Code*	*Description*
`Swapserv`	16	Client's swap server, IP.
`Rootpath`	17	Client's Root path, ASCII.
`ExtendP`	18	Extensions path, ASCII.
`IpFwdF`	19	IP Forwarding Enable / Disable, NUMBER.
`NLrouteF`	20	Non-local Source Routing, NUMBER.
`PFilter`	21	Policy Filter, IP,IP.
`MaxIpSiz`	22	Maximum datagram Reassembly Size, NUMBER.
`IpTTL`	23	Default IP Time to Live, (1=<x<=255), NUMBER.
`PathTO`	24	RFC-1191 Path MTU Aging Timeout, NUMBER.
`PathTbl`	25	RFC-1191 Path MTU Plateau Table, NUMBER.
`MTU`	26	Interface MTU, x>=68, NUMBER.
`SameMtuF`	27	All Subnets are Local, NUMBER.
`Broadcst`	28	Broadcast Address, IP.
`MaskDscF`	29	Perform Mask Discovery, NUMBER.
`MaskSupF`	30	Mask Supplier, NUMBER.
`RDiscvyF`	31	Perform Router Discovery, NUMBER.
`RSolictS`	32	Router Solicitation Address, IP.
`StaticRt`	33	Static Route, Double IP (network router).
`TrailerF`	34	Trailer Encapsulation, NUMBER.
`ArpTimeO`	35	ARP Cache Time out, NUMBER.

Symbol	*Code*	*Description*
EthEncap	36	Ethernet Encapsulation, NUMBER.
TcpTTL	37	TCP Default Time to Live, NUMBER.
TcpKaInt	38	TCP Keepalive Interval, NUMBER.
TcpKaGbF	39	TCP Keepalive Garbage, NUMBER.
NISdmain	40	NIS Domain name, ASCII.
NISservs	41	List of NIS servers, IP.
NTPservs	42	List of NTP servers, IP.
NetBNms	44	List of NetBIOS Name servers, IP.
NetBDsts	45	List of NetBIOS Distribution servers, IP.
NetBNdT	46	NetBIOS Node type (1=B-node, 2=P, 4=M, 8=H)
NetBScop	47	NetBIOS scope, ASCII.
XFontSrv	48	List of X Window Font servers, IP.
XDispMgr	49	List of X Window Display managers, IP.
LeaseTim	51	Lease Time Policy, (-1 = PERM), NUMBER.
Message	56	Message to be displayed on client, ASCII.
T1Time	58	Renewal (T1) time, NUMBER.
T2Time	59	Rebinding (T2) time, NUMBER.
NW_dmain	62	NetWare/IP Domain Name, ASCII.
NWIPOpts	63	NetWare/IP Options, OCTET (unknown type).
NIS+dom	64	NIS+ Domain name, ASCII.

Symbol	*Code*	*Description*
NIS+serv	65	NIS+ servers, IP.
TFTPsrvN	66	TFTP server hostname, ASCII.
OptBootF	67	Optional Bootfile path, ASCII.
MblIPAgt	68	Mobile IP Home Agent, IP.
SMTPserv	69	Simple Mail Transport Protocol Server, IP.
POP3serv	70	Post Office Protocol (POP3) Server, IP.
NNTPserv	71	Network News Transport Proto. (NNTP) Server, IP.
WWWservs	72	Default WorldWideWeb Server, IP.
Fingersv	73	Default Finger Server, IP.
IRCservs	74	Internet Relay Chat Server, IP.
STservs	75	StreetTalk Server, IP.
STDAservs	76	StreetTalk Directory Assist. Server, IP.
BootFile	N/A	File to Boot, ASCII.
BootPath	N/A	Boot path prefix to apply to client's requested boot file, ASCII.
BootSrvA	N/A	Boot Server, IP.
BootSrvN	N/A	Boot Server Hostname, ASCII.
EchoVC	N/A	Echo Vendor Class Identifier Flag, (Present=TRUE)
LeaseNeg	N/A	Lease is Negotiable Flag, (Present=TRUE)
Include	N/A	Include listed macro values in this macro.

EXAMPLES

EXAMPLE 1 A Sample dhcptab File

```
#
# Solaris-specific client vendor options. First define them, then use them
# in our Client Class macro definitions to establish proper context for each
# specific platform. Used to implement diskless boot of Solaris using DHCP
# as a configuration protocol.
#

# Root NFS mount options (mount_nfs(1M) form)
SrootOpt   s     Vendor=SUNW.Ultra-1 SUNW.Ultra-30 SUNW.i86pc,1,ASCII,1,0

# IP address of Root server.
SrootIP4   s     Vendor=SUNW.Ultra-1 SUNW.Ultra-30 SUNW.i86pc,2,IP,1,1

# Hostname of Root server.
SrootNM    s     Vendor=SUNW.Ultra-1 SUNW.Ultra-30 SUNW.i86pc,3,ASCII,1,0

# Pathname of Root directory.
SrootPTH   s     Vendor=SUNW.Ultra-1 SUNW.Ultra-30 SUNW.i86pc,4,ASCII,1,0

# IP address of Swap server.
SswapIP4   s     Vendor=SUNW.Ultra-1 SUNW.Ultra-30 SUNW.i86pc,5,IP,1,0

# Path to swapfile on swap server.
SswapPTH   s     Vendor=SUNW.Ultra-1 SUNW.Ultra-30 SUNW.i86pc,6,ASCII,1,0

# Option path of file to boot (e.g. /platform/sun4u/kernel/sparcv9/unix)
SbootFIL   s     Vendor=SUNW.Ultra-1 SUNW.Ultra-30 SUNW.i86pc,7,ASCII,1,0

# Posix 1003.1 timezone specification
Stz        s     Vendor=SUNW.Ultra-1 SUNW.Ultra-30 SUNW.i86pc,8,ASCII,1,0

# NFS read size used by standalone boot program when loading kernel.
SbootRS    s     Vendor=SUNW.Ultra-1 SUNW.Ultra-30 SUNW.i86pc,9,NUMBER,2,1

# IP address of Jumpstart Install server.
SinstIP4   s     Vendor=SUNW.Ultra-1 SUNW.Ultra-30 SUNW.i86pc,10,IP,1,1

# Name of Jumpstart Install server.
SinstNM    s     Vendor=SUNW.Ultra-1 SUNW.Ultra-30 SUNW.i86pc,11,ASCII,1,0

# Path to installation image on Install server.
SinstPTH   s     Vendor=SUNW.Ultra-1 SUNW.Ultra-30 SUNW.i86pc,12,ASCII,1,0

# ASCII <server name>:/path of sysid configuration file
SsysidCF   s     Vendor=SUNW.Ultra-1 SUNW.Ultra-30 SUNW.i86pc,13,ASCII,1,0

# ASCII <server name>:/path of JumpStart configuration file
SjumpsCF   s     Vendor=SUNW.Ultra-1 SUNW.Ultra-30 SUNW.i86pc,14,ASCII,1,0

# ASCII terminal type
Sterm      s     Vendor=SUNW.Ultra-1 SUNW.Ultra-30 SUNW.i86pc,15,ASCII,1,0

#
# Macro definitions
```

```
#
# Set the Locale. EST’s offset from GMT is -18000 seconds.
Locale          m     :UTCoffst=-18000:
#
# Define all Solaris-generic options under this macro.
Solaris         m      :SrootIP4=172.21.0.2:SrootNM="test-172-21-0-0-2": \
                       :SinstIP4=172.21.0.2:SinstNM="test-172-21-0-0-2": \
                       :Sterm="xterm":
#
# Define all sparc-platform specific options under this macro.
sparc           m       \
:SrootPTH="/export/s28/base.s28s_wos/latest/Solaris_2.8/Tools/Boot": \
                       :SinstPTH="/export/s28/base.s28s_wos/latest":
#
# Define all sun4m architecture-specific options under this macro. Note how
# we include the Solaris and sparc generic information by include the
# appropriate macros in this definition.
sun4m           m      :Include=Solaris:Include=sparc: \
                       :SbootFIL="/platform/sun4m/kernel/unix":
#
# Define all sun4u architecture-specific options under this macro.
sun4u           m      :Include=Solaris:Include=sparc:
#
# Solaris on Intel platform-specific parameters are under this macro.
i86             m      :Include=Solaris: \
:SrootPTH="/export/s28/base.s28x_wos/latest/Solaris_2.8/Tools/Boot": \
                       :SinstPTH="/export/s28/base.s28x_wos/latest": \
                       :SbootFIL="/platform/i86pc/kernel/unix":
#
# Solaris on Intel machines are identified by the "SUNW.i86pc" class. All
# clients identifying themselves as members of this class will see these
# parameters.
SUNW.i86pc     m     :Include=i86:
#
# Ultra-1 platforms identify themselves as part of the "SUNW.Ultra-1" class.
# By default, we boot these machines in 32bit mode. All clients identifying
# themselves as members of this class will see these parameters.
SUNW.Ultra-1     m     :SbootFIL="/platform/sun4u/kernel/unix": \
                       :Include=sun4u:
#
# Ultra-30 platforms identify themselves as part of the "SUNW.Ultra-30" class.
# By default, we will boot these machines in 64bit mode. All clients
# identifying themselves as members of this class will see these parameters.
SUNW.Ultra-30     m     :SbootFIL="/platform/sun4u/kernel/sparcv9/unix": \
                        :Include=sun4u:
#
# Macros named using a client’s subnet IP address are automatically consulted
# by the DHCP server. Thus, all clients on the 172.20.64.64 network will see
# these options. Thus it makes sense to associate all parameters specific to
# a network with its macro. Note that it is important to keep the netmasks(4)
# table up to date with respect to your network topology in order for the
# DHCP server macro selection process to work correctly.
#
172.20.64.64      m     :Broadcst=172.20.64.127:Subnet=255.255.255.192: \
                        :Router=172.20.64.65:BootSrvA=172.21.0.2:
```

```
172.20.64.0       m     :Subnet=255.255.255.192: \
                        :Router=172.20.64.2 172.20.64.1:Broadcst=172.20.64.63: \
                        :BootSrvA=172.21.0.2:
172.20.64.128     m     :Subnet=255.255.255.128:Router=172.20.64.129: \
                        :Broadcst=172.20.64.255:BootSrvA=172.21.0.2:
172.21.0.0        m     :Subnet=255.255.0.0:Router=172.21.0.1: \
                        :Broadcst=172.21.255.255:BootSrvA=172.21.0.2:
192.168.208.0     m     :Subnet=255.255.248.0:Router=192.168.208.1: \
                        :Broadcst=192.168.215.255:BootSrvA=172.21.0.2:
172.22.0.0        m     :Broadcst=172.22.255.255:Subnet=255.255.0.0:MTU=4352: \
                        :Router=172.22.0.1:BootSrvA=172.22.0.1: \
                        :NIS+dom="nis+.labtest.dhcp":NIS+serv=172.21.0.2:
#
# We use a macro named after the server’s hostname to group parameters related
# to the services exported by this server. Here we set the lease policy, as
# well as automatically return a client’s hostname by consulting the name
# service.
test-172-21-0-0-2 m     :Include=Locale:Timeserv=172.21.0.2: \
                        :LeaseTim=3600:LeaseNeg:Hostname: \
                        :DNSdmain="lab.test.dhcp":DNSserv=172.22.0.7:
#
# This macro’s name is a client’s client identifier. Its options will be
# combined with those of the Client class macro, network macro, and server
# macro. Regardless of where this client appears in the network topology
# served by this dhcp service, these parameters will follow it!
010800207E8A02   m       :Impress=172.22.255.27:
```

FILES

`/var/dhcp/dhcptab` file or NIS+ table.

SEE ALSO

`dhcpconfig`(1M), `dhcpmgr`(1M), `dhtadm`(1M), `in.dhcpd`(1M), `dhcp_network`(4)

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

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

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

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

NAME

dialups – list of terminal devices requiring a dial-up password

SYNOPSIS

`/etc/dialups`

DESCRIPTION

`dialups` is an ASCII file which contains a list of terminal devices that require a dial-up password. A dial-up password is an additional password required of users who access the computer through a modem or dial-up port. The correct password must be entered before the user is granted access to the computer. The set of ports that require a dial-up password are listed in the `dialups` file.

Each entry in the `dialups` file is a single line of the form:

terminal-device

where

terminal-device The full path name of the terminal device that will require a dial-up password for users accessing the computer through a modem or dial-up port.

The `dialups` file should be owned by the `root` user and the `root` group. The file should have read and write permissions for the owner (`root`) only.

EXAMPLES

EXAMPLE 1 A sample `dialups` file.

Here is a sample `dialups` file:

```
/dev/term/a
/dev/term/b
/dev/term/c
```

FILES

`/etc/d_passwd`	dial-up password file
`/etc/dialups`	list of dial-up ports requiring dial-up passwords

SEE ALSO

`d_passwd`(4)

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 {
        ulong_t  d_ino;                  /* inode number of entry */
        ushort_t d_reclen;               /* length of this record */
        ushort_t d_namlen;               /* length of string in d_name */
        char     d_name[MAXNAMLEN + 1];  /* maximum name length */
};
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Stability Level	Unstable

SEE ALSO

fs_ufs(4) , attributes(5)

NAME

d_passwd – dial-up password file

SYNOPSIS

`/etc/d_passwd`

DESCRIPTION

A dial-up password is an additional password required of users who access the computer through a modem or dial-up port. The correct password must be entered before the user is granted access to the computer.

`d_passwd` is an ASCII file which contains a list of executable programs (typically shells) that require a dial-up password and the associated encrypted passwords. When a user attempts to log in on any of the ports listed in the `dialups` file (see `dialups`(4)), the login program looks at the user's login entry stored in the `passwd` file (see `passwd`(4)), and compares the login shell field to the entries in `d_passwd`. These entries determine whether the user will be required to supply a dial-up password.

Each entry in `d_passwd` is a single line of the form:

login-shell`:`*password*`:`

where

login-shell — The name of the login program that will require an additional dial-up password.

password — A 13-character encrypted password. Users accessing the computer through a dial-up port or modem using *login-shell* will be required to enter this password before gaining access to the computer.

`d_passwd` should be owned by the `root` user and the `root` group. The file should have read and write permissions for the owner (`root`) only.

If the user's login program in the `passwd` file is not found in `d_passwd` or if the login shell field in `passwd` is empty, the user must supply the default password. The default password is the entry for `/usr/bin/sh`. If `d_passwd` has no entry for `/usr/bin/sh`, then those users whose login shell field in `passwd` is empty or does not match any entry in `d_passwd` will not be prompted for a dial-up password.

Dial-up logins are disabled if `d_passwd` has only the following entry:

```
/usr/bin/sh:*:
```

EXAMPLES

EXAMPLE 1 Sample d_passwd file.

Here is a sample d_passwd file:

```
/usr/lib/uucp/uucico:q.mJzTnu8icF0:
/usr/bin/csh:6k/7KCFRPNVXg:
/usr/bin/ksh:9df/FDf.4jkRt:
/usr/bin/sh:41FuGVzGcDJlw:
```

Generating An Encrypted Password

The passwd (see passwd(1)) utility can be used to generate the encrypted password for each login program. passwd generates encrypted passwords for users and places the password in the shadow (see shadow(4)) file. Passwords for the d_passwd file will need to be generated by first adding a temporary user id using useradd (see useradd(1M)), and then using passwd(1) to generate the desired password in the shadow file. Once the encrypted version of the password has been created, it can be copied to the d_passwd file.

For example:

1. Type useradd tempuser and press Return. This creates a user named tempuser.
2. Type passwd tempuser and press Return. This creates an encrypted password for tempuser and places it in the shadow file.
3. Find the entry for tempuser in the shadow file and copy the encrypted password to the desired entry in the d_passwd file.
4. Type userdel tempuser and press Return to delete tempuser.

These steps must be executed as the root user.

FILES

/etc/d_passwd	dial-up password file
/etc/dialups	list of dial-up ports requiring dial-up passwords
/etc/passwd	password file
/etc/shadow	shadow password file

SEE ALSO

passwd(1), useradd(1M), dialups(4), passwd(4), shadow(4)

WARNINGS

When creating a new dial-up password, be sure to remain logged in on at least one terminal while testing the new password. This ensures that there is an available terminal from which you can correct any mistakes that were made when the new password was added.

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_get_int`(9F) and `ddi_prop_lookup`(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 can be either a simple nexus driver name to match all instances of that parent/node, or the parent name can be a specific full pathname, beginning with a slash (`/`) character, identifying a specific instance of a parent bus.

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 retrieved using the DDI property interfaces (for example, `ddi_prop_get_int`(9F) and `ddi_prop_lookup`(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_prop_lookup_int_array`(9F).

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

EXAMPLE 1 Configuration file for a PCI bus frame buffer.

The following is an example of a configuration file called `ACME,simple.conf` for a PCI bus frame buffer called `ACME,simple`.

```
#
# Copyright (c) 1993, by ACME Fictitious Devices, Inc.
#
#ident  "@(#)ACME,simple.conf   1.3     1999/09/09"

name="ACME,simple" class="pci" unit-address="3,1"
        debug-mode=12;
```

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

CODE EXAMPLE 1 Configuration file for a pseudo device driver

The following is an example of 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 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_prop_get_int`(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

`pci`(4), `pseudo`(4), `sbus`(4), `scsi`(4), `pci`(4), `probe`(9E), `ddi_getlongprop`(9F), `ddi_getprop`(9F), `ddi_getproplen`(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, pref, variables – user-preference variables files for AT&T FACE

SYNOPSIS

`$HOME/pref/.environ`

`$HOME/pref/.variables`

`$HOME/FILECABINET/.pref`

`$HOME/WASTEBASKET/.pref`

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* .

.environ Variables 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 with which files will be created.

.pref Variables

Variables found in `.pref` are the following:

SORTMODE Contains the same values as the SORTMODE variable described in `.environ` above.

DISPMODE Contains the same values as the DISPLAYMODE variable described in `.environ` above.

.variable Variables

Variables found in `.variables` include:

EDITOR Default editor.

PS1 Shell prompt.

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`(3SOCKET) 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`(3SOCKET), `hosts`(4), `nsswitch.conf`(4)

NAME

exec_attr – execution profiles database

SYNOPSIS

/etc/security/exec_attr

DESCRIPTION

/etc/security/exec_attr is a local database that specifies the execution attributes associated with profiles. The exec_attr file can be used with other sources for execution profiles, including the exec_attr NIS map and NIS+ table. Programs use the getexecattr(3SECDB) routines to access this information.

The search order for multiple execution profile sources is specified in the /etc/nsswitch.conf file, as described in the nsswitch.conf(4) man page. The search order follows the entry for prof_attr(4).

A profile is a logical grouping of authorizations and commands that is interpreted by a profile shell to form a secure execution environment. The shells that interpret profiles are pfcsh, pfksh, and pfsh. See the pfsh(1) man page. Each user's account is assigned zero or more profiles in the user_attr(4) database file.

Each entry in the exec_attr database consists of one line of text containing seven fields separated by colons (:). Line continuations using the backslash (\) character are permitted. The basic format of each entry is:

name:*policy*:*type*:*res1*:*res2*:*id*:*attr*

name The name of the profile. Profile names are case-sensitive.

policy The policy that is associated with the profile entry. The only valid *policy* is suser.

type The type of object defined in the profile. The only valid type is cmd.

res1 Reserved for future use.

res2 Reserved for future use.

id A string that uniquely identifies the object described by the profile. For a profile of type cmd, the id is either the full path to the command or the asterisk (*) symbol, which is used to allow all commands. An asterisk that replaces the filename component in a pathname indicates all files in a particular directory. To specify arguments, the pathname should point to a shell script written to execute the command with the desired arguments.

attr — An optional list of semicolon-separated (`;`) key-value pairs that describe the security attributes to apply to the object upon execution. Zero or more keys may be specified. The list of valid key words depends on the policy enforced. The following key words are valid: `euid`, `uid`, `egid`, and `gid`.

`euid` and `uid` contain a single user name or a numeric user ID. Commands designated with `euid` run with the effective UID indicated, which is similar to setting the setuid bit on an executable file. Commands designated with `uid` run with both the real and effective UIDs. Setting `uid` may be more appropriate than setting the `euid` on privileged shell scripts.

`egid` and `gid` contain a single group name or a numeric group ID. Commands designated with `egid` run with the effective GID indicated, which is similar to setting the setgid bit on a file. Commands designated with `gid` run with both the real and effective GIDs. Setting `gid` may be more appropriate than setting `guid` on privileged shell scripts.

EXAMPLES

EXAMPLE 1 Using effective user and group IDs

The following example shows the `audit` command specified in the Audit Control profile to execute with an effective user ID of root (`0`) and effective group ID of bin (`3`):

```
Audit Control:suser:cmd:::/etc/init.d/audit:euid=0;egid=3
```

FILES

`/etc/nsswitch.conf`

`/etc/user_attr`

`/etc/security/exec_attr`

CAVEATS

When deciding which authorization source to use (see DESCRIPTION), keep in mind that NIS+ provides stronger authentication than NIS.

Because the list of legal keys is likely to expand, any code that parses this database must be written to ignore unknown key-value pairs without error. When any new keywords are created, the names should be prefixed with a unique string, such as the company's stock symbol, to avoid potential naming conflicts.

The following characters are used in describing the database format and must be escaped with a backslash if used as data: colon (`:`), semicolon (`;`), equals (`=`), and backslash (`\`).

SEE ALSO

auths(1), profiles(1), roles(1), makedbm(1M), getauthattr(3SECDB), getauusernam(3BSM), getexecattr(3SECDB), getprofattr(3SECDB), getuserattr(3SECDB), kva_match(3SECDB), auth_attr(4), prof_attr(4), user_attr(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

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 compatible 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 single 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/cntndnsn`; 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 environment 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 (also known as 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 partition 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

EXAMPLE 1 A sample `disk_type` and `partition`.

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) *System Administration Guide, Volume 1*

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

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

`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. The FSFIX state indicates that this fs is mounted, contains inconsistent file system data and is being repaired by `fsck` . The FSBAD state indicates that this file system contains inconsistent file system data. 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 `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 `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 `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 `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> .

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Stability Level	Unstable

SEE ALSO

fsck_ufs(1M) , mkfs_ufs(1M) , tunefs(1M) , mount(2) , attributes(5)

NAME

ftpusers – file listing users to be disallowed ftp login privileges

SYNOPSIS

`/etc/ftpusers`

DESCRIPTION

The `/etc/ftpusers` is an ASCII file that lists users for whom `ftp` login privileges are disallowed. Each `ftpuser` entry is a single line of the form:

```
name
```

where `name` is the user's login name.

The `ftp` server, `in.ftpd`(1M), reads the `ftpusers` file. If the login name of the user matches one of the entries listed, it rejects the login session and sends the `Login incorrect` and `Login failed` error messages.

The `ftpusers` file has the following default configuration entries:

```
root
daemon
bin
sys
adm
lp
uccp
nuucp
listen
nobody
noaccess
nobody4
```

These entries match the default instantiated entries from `passwd`(4). The list of default entries typically contains the superuser `root` and other administrative and system application identities.

The root entry is included in `/etc/ftpusers` as a security measure since the default policy is to disallow remote logins for this identity. This policy is also set in the the default value of the `CONSOLE` entry in the `/etc/default/login` file. See `login`(1). If you allow `root` login privileges by deleting the root entry in `/etc/ftpusers`, you should also should modify the security policy in `/etc/default/login` to reflect the site security policy for remote login access by `root`.

Other default entries are administrative identities that are typically assumed by system applications but never used for local or remote login, for example `sys` and `nobody`. Since these entries do not have a valid password field instantiated in `shadow`(4), no login can be performed.

If a site adds similar administrative or system application identities in passwd(4) and shadow(4), for example, majordomo, the site should consider including them in /etc/ftpusers for a consistent security policy.

FILES

/etc/ftpusers

/etc/default/login

/etc/passwd

/etc/shadow

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWftpr

SEE ALSO

login(1), in.ftpd(1M), passwd(4), shadow(4), attributes(5), environ(5)

NAME

geniconvtbl – geniconvtbl input file format

DESCRIPTION

An input file to geniconvtbl is an ASCII text file that contains an iconv code conversion definition from one codeset to another codeset.

The geniconvtbl utility accepts the code conversion definition file(s) and writes code conversion binary table file(s) that can be used in iconv(1) and iconv(3C) to support user-defined code conversions. See iconv(1) and iconv(3C)for more detail on the iconv code conversion and geniconvtbl(1) for more detail on the utility.

The Lexical Conventions

The following lexical conventions are used in the iconv code conversion definition:

CONVERSION_NAME
: A string of characters representing the name of the iconv code conversion. The iconv code conversion name should start with one or more printable ASCII characters followed by a percentage character '%' followed by another one or more of printable ASCII characters. Examples: `ISO8859-1%ASCII`, `646%eucJP`, `CP_939%ASCII`.

NAME
: A string of characters starts with any one of the ASCII alphabet characters or the underscore character, '_', followed by one or more ASCII alphanumeric characters and underscore character, '_'. Examples: `_a1`, `ABC_codeset`, `K1`.

HEXADECIMAL
: A hexadecimal number. The hexadecimal representation consists of an escape character, '0' followed by the constant 'x' or 'X' and one or more hexadecimal digits. Examples: `0x0`, `0x1`, `0x1a`, `0X1A`, `0x1B3`.

DECIMAL
: A decimal number, represented by one or more decimal digits. Examples: `0`, `123`, `2165`.

Each comment starts with '//' ends at the end of the line.

The following keywords are reserved:

automatic	between	binary
break	condition	default
dense	direction	discard
else	error	escapeseq

false	if	index
init	input	inputsize
map	maptype	no_change_copy
operation	output	output_byte_length
outputsize	printchr	printhd
printint	reset	return
true		

Additionally, the following symbols are also reserved as tokens:

{ } [ ] ( ) ; , ...

The precedence and associativity

The following table shows the precedence and associativity of the operators from lower precedence at the top to higher precedence at the bottom of the table allowed in the iconv code conversion definition:

```
Operator (Symbol)                              Associativity
-------------------------------------------------
Assignment (=)                                 Right
-------------------------------------------------
Logical OR (||)                                Left
-------------------------------------------------
Logical AND (&&)                               Left
-------------------------------------------------
Bitwise OR (|)                                 Left
-------------------------------------------------
Exclusive OR (^)                               Left
-------------------------------------------------
Bitwise AND (&)                                Left
-------------------------------------------------
Equal-to (= =),                                Left
   Inequality (!=)
-------------------------------------------------
Less-than (<),                                 Left
   Less-than-or-equal-to (<=),
   Greater-than (>),
   Greater-than-or-equal-to (>=)
-------------------------------------------------
Left-shift (<<),                               Left
   Right-shift (>>)
```

(continued)

(Continuation)

```
-------------------------------------------------
Addition (+),                               Left
   Subtraction (-)
-------------------------------------------------
Multiplication (*),                         Left
   Division (/),
   Remainder (%)
--------------------------------------------------
Logical negation (!),                       Right
   Bitwise complement (~),
   Unary minus (-)
--------------------------------------------------
```

The Syntax

Each iconv code conversion definition starts with `CONVERSION_NAME` followed by one or more semi-colon separated code conversion definition elements:

```
// a US-ASCII to ISO8859-1 iconv code conversion example:
US-ASCII%ISO8859-1 {

    // one or more code conversion definition elements here.

    :
    :

}
```

Each code conversion definition element can be any one of the following elements:

direction
condition
operation
map

To have a meaningful code conversion, there should be at least one direction, operation, or map element in the iconv code conversion definition.

The direction element contains one or more semi-colon separated condition-action pairs that direct the code conversion:

```
direction For_US-ASCII_2_ISO8859-1 {

    // one or more condition-action pairs here.
    :
```

```
      :

  }
```

Each condition-action pair contains a conditional code conversion that consists of a condition element and an action element.

condition action

If the pre-defined condition is met, the corresponding action is executed. If there is no pre-defined condition met, `iconv`(3C) will return `-1` with errno set to `EILSEQ`. The condition can be a condition element, a name to a pre-defined condition element, or a condition literal value, true. The ’true’ condition literal value always yields success and thus the corresponding action is always executed. The action also can be an action element or a name to a pre-defined action element.

The condition element specifies one or more condition expression elements. Since each condition element can have a name and also can exist stand-alone, a pre-defined condition element can be referenced by the name at any action pairs later. To be used in that way, the corresponding condition element should be defined beforehand:

```
  condition For_US-ASCII_2_ISO8859-1 {

      // one or more condition expression elements here.
      :
      :

  }
```

The name of the condition element in the above example is `For_US-ASCII_2_ISO8859-1`. Each condition element can have one or more condition expression elements. If there are more than one condition expression elements, the condition expression elements are checked from top to bottom to see if any one of the condition expression elements will yield a true. Any one of the following can be a condition expression element:

between
escapeseq
expression

The between condition expression element defines one or more comma-separated ranges:

```
between 0x0...0x1f, 0x7f...0x9f ;
between 0xa1a1...0xfefe ;
```

In the first expression in the example above, the covered ranges are `0x0` to `0x1f` and `0x7f` to `0x9f` inclusively. In the second expression, the covered range is the range whose first byte is `0xa1` to `0xfe` and whose second byte is between `0xa1` to `0xfe`. This means that the range is defined by each byte. In this case, the sequence `0xa280` does not meet the range.

The escapeseq condition expression element defines an equal-to condition for one or more comma-separated escape sequence designators:

```
// ESC $ ) C sequence:
escapeseq 0x1b242943;

// ESC $ ) C sequence or ShiftOut (SO) control character code, 0x0e:
escapeseq 0x1b242943, 0x0e;
```

The expression can be any one of the following and can be surrounded by a pair of parentheses, '(' and ')':

```
// HEXADECIMAL:
0xa1a1

// DECIMAL
12

// A boolean value, true:
true

// A boolean value, false:
false

// Addition expression:
1 + 2

// Subtraction expression:
10 - 3

// Multiplication expression:
0x20 * 10

// Division expression:
20 / 10

// Remainder expression:
17 % 3
```

```
// Left-shift expression:
1 << 4

// Right-shift expression:
0xa1 >> 2

// Bitwise OR expression:
0x2121 | 0x8080

// Exclusive OR expression:
0xa1a1 ^ 0x8080

// Bitwise AND expression:
0xa1 & 0x80

// Equal-to expression:
0x10 == 16

// Inequality expression:
0x10 != 10

// Less-than expression:
0x20 < 25

// Less-than-or-equal-to expression:
10 <= 0x10

// Bigger-than expression:
0x10 > 12

// Bigger-than-or-equal-to expression:
0x10 >= 0xa

// Logical OR expression:
0x10 || false

// Logical AND expression:
0x10 && false

// Logical negation expression:
! false

// Bitwise complement expression:
~0

// Unary minus expression:
-123
```

There is a single type available in this expression: integer. The boolean values are two special cases of integer values. The 'true' boolean value's integer value is `1` and the 'false' boolean value's integer value is `0`. Also, any integer value other than `0` is a true boolean value. Consequently, the integer value `0` is the false boolean value. Any boolean expression yields integer value `1` for true and integer value `0` for false as the result.

Any literal value shown at the above expression examples as operands, that is, DECIMAL, HEXADECIMAL, and boolean values, can be replaced with another expression. There are a few other special operands that you can use as well in the expressions: '`input`', '`inputsize`', '`outputsize`', and variables. `input` is a keyword pointing to the current input buffer. `inputsize` is a keyword pointing to the current input buffer size in bytes. `outputsize` is a keyword pointing to the current output buffer size in bytes. The NAME lexical convention is used to name a variable. The initial value of a variable is `0`. The following expressions are allowed with the special operands:

```
// Pointer to the third byte value of the current input buffer:
input[2]

// Equal-to expression with the 'input':
input == 0x8020

// Alternative way to write the above expression:
0x8020 == input

// The size of the current input buffer size:
inputsize

// The size of the current output buffer size:
outputsize

// A variable:
saved_second_byte

// Assignment expression with the variable:
saved_second_byte = input[1]
```

The `input` keyword without index value can be used only with the equal-to operator, '=='. When used in that way, the current input buffer is consecutively compared with another operand byte by byte. An expression can be another operand. If the `input` keyword is used with an index value *n*, it is a pointer to the (*n*+1)th byte from the beginning of the current input buffer. An expression can be the index. Only a variable can be placed on the left hand side of an assignment expression.

The action element specifies an action for a condition and can be any one of the following elements:

direction
operation

(continued)

(Continuation)

map

The operation element specifies one or more operation expression elements:

```
operation For_US-ASCII_2_ISO8859-1 {

    // one or more operation expression element definitions here.
    :
    :

}
```

If the name of the operation element, in the case of the above example, `For_US-ASCII_2_ISO8859-1`, is either `init` or `reset`, it defines the initial operation and the reset operation of the iconv code conversion:

```
// The initial operation element:
operation init {

    // one or more operation expression element definitions here.
    :
    :

}

// The reset operation element:
operation reset {

    // one or more operation expression element definitions here.
    :
    :

}
```

The initial operation element defines the operations that need to be performed in the beginning of the iconv code conversion. The reset operation element defines the operations that need to be performed when a user of the iconv(3) function requests a state reset of the iconv code conversion. For more detail on the state reset, refer to `iconv`(3C).

The operation expression can be any one of the following three different expressions and each operation expression should be separated by an ending semicolon:

if-else operation expression
output operation expression
control operation expression

The if-else operation expression makes a selection depend on the boolean expression result. If the boolean expression result is true, the true task that follows the 'if' is executed. If the boolean expression yields false and if a false task is supplied, the false task that follows the 'else' is executed. There are three different kinds of if-else operation expressions:

```
// The if-else operation expression with only true task:
if (expression) {

    // one or more operation expression element definitions here.
    :
    :

}

// The if-else operation expression with both true and false
// tasks:
if (expression) {

    // one or more operation expression element definitions here.
    :
    :

} else {

    // one or more operation expression element definitions here.
    :
    :

}

// The if-else operation expression with true task and
// another if-else operation expression as the false task:
if (expression) {

    // one or more operation expression element definitions here.
    :
    :

} else if (expression) {

    // one or more operation expression element definitions here.
    :
    :

} else {
```

```
     // one or more operation expression element definitions here.
     :
     :

}
```

The last if-else operation expression can have another if-else operation expression as the false task. The other if-else operation expression can be any one of above three if-else operation expressions.

The output operation expression saves the right hand side expression result to the output buffer:

```
// Save 0x8080 at the output buffer:
output = 0x8080;
```

If the size of the output buffer left is smaller than the necessary output buffer size resulting from the right hand side expression, the iconv code conversion will stop with `E2BIG` errno and `(size_t)-1` return value to indicate that the code conversion needs more output buffer to complete. Any expression can be used for the right hand side expression. The output buffer pointer will automatically move forward appropriately once the operation is executed.

The control operation expression can be any one of the following expressions:

```
// Return (size_t)-1 as the return value with an EINVAL errno:
error;

// Return (size_t)-1 as the return value with an EBADF errno:
error 9;

// Discard input buffer byte operation. This discards a byte from
// the current input buffer and move the input buffer pointer to
// the 2'nd byte of the input buffer:
discard;

// Discard input buffer byte operation. This discards
// 10 bytes from the current input buffer and move the input
// buffer pointer to the 11'th byte of the input buffer:
discard 10;

// Return operation. This stops the execution of the current
// operation:
return;

// Operation execution operation. This executes the init
// operation defined and sets all variables to zero:
operation init;
```

```
// Operation execution operation. This executes the reset
// operation defined and sets all variables to zero:
operation reset;

// Operation execution operation. This executes an operation
// defined and named 'ISO8859_1_to_ISO8859_2':
operation ISO8859_1_to_ISO8859_2;

// Direction operation. This executes a direction defined and
// named 'ISO8859_1_to_KOI8_R:
direction ISO8859_1_to_KOI8_R;

// Map execution operation. This executes a mapping defined
// and named 'Map_ISO8859_1_to_US_ASCII':
map Map_ISO8859_1_to_US_ASCII;

// Map execution operation. This executes a mapping defined
// and named 'Map_ISO8859_1_to_US_ASCII' after discarding
// 10 input buffer bytes:
map Map_ISO8859_1_to_US_ASCII 10;
```

In case of init and reset operations, if there is no pre-defined init and / or reset operations in the iconv code conversions, only system-defined internal init and reset operations will be executed. The execution of the system-defined internal init and reset operations will clear the system-maintained internal state.

There are three special operators that can be used in the operation:

```
printchr expression;
printhd expression;
printint expression;
```

The above three operators will print out the given expression as a character, a hexadecimal number, and a decimal number, respectively, at the standard error stream. These three operators are for debugging purposes only and should be removed from the final version of the iconv code conversion definition file.

In addition to the above operations, any valid expression separated by a semi-colon can be an operation, including an empty operation, denoted by a semi-colon alone as an operation.

The map element specifies a direct code conversion mapping by using one or more map pairs. When used, usually many map pairs are used to represent an iconv code conversion definition:

```
map For_US-ASCII_2_ISO8859-1 {
```

```
      // one or more map pairs here
      :
      :

}
```

Each map element also can have one or two comma-separated map attribute elements like the following examples:

```
// Map with densely encoded mapping table map type:
map maptype = dense {

      // one or more map pairs here
      :
      :

}

// Map with hash mapping table map type with hash factor 10.
// Only hash mapping table map type can have hash factor. If
// the hash factor is specified with other map types, it will be
// ignored.
map maptype = hash : 10 {

    // one or more map pairs here.
    :
    :

}

// Map with binary search tree based mapping table map type:
map maptype = binary {

    // one more more map pairs here.
    :
    :
}

// Map with index table based mapping table map type:
map maptype = index {

    // one or more map pairs here.
    :
    :

}

// Map with automatic mapping table map type. If defined,
// system will assign the best possible map type.
map maptype = automatic {

    // one or more map pairs here.
    :
```

```
    :

}

// Map with output_byte_length limit set to 2.
map output_byte_length = 2 {

    // one or more map pairs here.
    :
    :

}

// Map with densely encoded mapping table map type and
// output_bute_length limit set to 2:
map maptype = dense, output_byte_length = 2 {

   // one or more map pairs here.
    :
    :

}
```

If no maptype is defined, automatic is assumed. If no output_byte_length is defined, the system figures out the maximum possible output byte length for the mapping by scanning all the possible output values in the mappings. If the actual output byte length scanned is bigger than the defined output_byte_length, the `geniconvtbl` utility issues an error and stops generating the code conversion binary table(s).

The following are allowed map pairs:

```
// Single mapping. This maps an input character denoted by
// the code value 0x20 to an output character value 0x21:
0x20        0x21

// Multiple mapping. This maps 128 input characters to 128
// output characters. In this mapping, 0x0 maps to 0x10, 0x1 maps
// to 0x11, 0x2 maps to 0x12, ..., and, 0x7f maps to 0x8f:
0x0...0x7f  0x10

// Default mapping. If specified, every undefined input character
// in this mapping will be converted to a specified character
// (in the following case, a character with code value of 0x3f):
default     0x3f;

// Default mapping. If specified, every undefined input character
// in this mapping will not be converted but directly copied to
// the output buffer:
default     no_change_copy;

// Error mapping. If specified, during the code conversion,
```

```
// if input buffer contains the byte value, in this case, 0x80,
// the iconv(3) will stop and return (size_t)-1 as the return
// value with EILSEQ set to the errno:
0x80        error;
```

If no default mapping is specified, every undefined input character in the mapping will be treated as an error mapping. and thus the `iconv`(3C) will stop the code conversion and return `(size_t)-1` as the return value with `EILSEQ` set to the errno.

The syntax of the iconv code conversion definition in extended BNF is illustrated below:

```
iconv_conversion_definition
        : CONVERSION_NAME '{' definition_element_list '}'
        ;

definition_element_list
        : definition_element ';'
        | definition_element_list definition_element ';'
        ;

definition_element
        : direction
        | condition
        | operation
        | map
        ;

direction
        : 'direction' NAME '{' direction_unit_list '}'
        | 'direction' '{' direction_unit_list '}'
        ;
direction_unit_list
        : direction_unit
        | direction_unit_list direction_unit
        ;

direction_unit
        : condition action ';'
        | condition NAME ';'
        | NAME action ';'
        | NAME NAME ';'
        | 'true' action ';'
        | 'true' NAME ';'
        ;

action
        : direction
        | map
        | operation
        ;
```

```
condition
        : 'condition' NAME '{' condition_list '}'
        | 'condition' '{' condition_list '}'
        ;

condition_list
        : condition_expr ';'
        | condition_list condition_expr ';'
        ;

condition_expr
        : 'between' range_list
        | expr
        | 'escapeseq' escseq_list ';'
        ;

range_list
        : range_pair
        | range_list ',' range_pair
        ;

range_pair
        : HEXADECIMAL '...' HEXADECIMAL
        ;

escseq_list
        : escseq
        | escseq_list ',' escseq
        ;

escseq  : HEXADECIMAL
        ;

map     : 'map' NAME '{' map_list '}'
        | 'map' '{' map_list '}'
        | 'map' NAME map_attribute '{' map_list '}'
        | 'map' map_attribute '{' map_list '}'
        ;

map_attribute
        : map_type ',' 'output_byte_length' '=' DECIMAL
        | map_type
        | 'output_byte_length' '=' DECIMAL ',' map_type
        | 'output_byte_length' '=' DECIMAL
        ;

map_type: 'maptype' '=' map_type_name : DECIMAL
        | 'maptype' '=' map_type_name
        ;

map_type_name
        : 'automatic'
        | 'index'
        | 'hash'
```

```
        | 'binary'
        | 'dense'
        ;

map_list
        : map_pair
        | map_list map_pair
        ;
map_pair
        : HEXADECIMAL HEXADECIMAL
        | HEXADECIMAL '...' HEXADECIMAL HEXADECIMAL
        | 'default'  HEXADECIMAL
        | 'default'  'no_change_copy'
        | HEXADECIMAL 'error'
        ;

operation
        : 'operation' NAME '{' op_list '}'
        | 'operation' '{' op_list '}'
        | 'operation' 'init' '{' op_list '}'
        | 'operation' 'reset' '{' op_list '}'
        ;

op_list : op_unit
        | op_list op_unit
        ;

op_unit : ';'
        | expr ';'
        | 'error' ';'
        | 'error' expr ';'
        | 'discard' ';'
        | 'discard' expr ';'
        | 'output' '=' expr ';'
        | 'direction' NAME ';'
        | 'operation' NAME ';'
        | 'operation' 'init' ';'
        | 'operation' 'reset' ';'
        | 'map' NAME ';'
        | 'map' NAME expr ';'
        | op_if_else
        | 'return' ';'
        | 'printchr' expr ';'
        | 'printhd' expr ';'
        | 'printint' expr ';'
        ;

op_if_else
        : 'if' '(' expr ')' '{' op_list '}'
        | 'if' '(' expr ')' '{' op_list '}' 'else' op_if_else
        | 'if' '(' expr ')' '{' op_list '}' 'else' '{' op_list '}'
        ;

expr    : '(' expr ')'
        | NAME
```

```
            | HEXADECIMAL
            | DECIMAL
            | 'input' '[' expr ']'
            | 'outputsize'
            | 'inputsize'
            | 'true'
            | 'false'
            | 'input' '==' expr
            | expr '==' 'input'
            | '!' expr
            | '~' expr
            | '-' expr
            | expr '+' expr
            | expr '-' expr
            | expr '*' expr
            | expr '/' expr
            | expr '%' expr
            | expr '<<' expr
            | expr '>>' expr
            | expr '|' expr
            | expr '^' expr
            | expr '&' expr
            | expr '==' expr
            | expr '!=' expr
            | expr '>'  expr
            | expr '>='  expr
            | expr '<'  expr
            | expr '<='  expr
            | NAME '=' expr
            | expr '||' expr
            | expr '&&' expr
            ;
```

EXAMPLES

EXAMPLE 1 Code conversion from ISO8859-1 to ISO646

```
ISO8859-1%ISO646 {
          // Use dense-encoded internal data structure.
          map maptype = dense {
                  default         0x3f
                  0x0...0x7f      0x0
          };
  }
```

EXAMPLE 2 Code conversion from eucJP to ISO-2022-JP

```
// Iconv code conversion from eucJP to ISO-2022-JP

        #include <sys/errno.h>

        eucJP%ISO-2022-JP {
             operation init {
                 codesetnum = 0;
             };
```

```
operation reset {
    if (codesetnum != 0) {
         // Emit state reset sequence, ESC ( J, for
         // ISO-2022-JP.
         output = 0x1b284a;
    }
    operation init;
};

direction {
    condition {             // JIS X 0201 Latin (ASCII)
         between 0x00...0x7f;
    } operation {
         if (codesetnum != 0) {
              // We will emit four bytes.
              if (outputsize <= 3) {
                      error E2BIG;
              }
              // Emit state reset sequence, ESC ( J.
              output = 0x1b284a;
              codesetnum = 0;
         } else {
              if (outputsize <= 0) {
                      error E2BIG;
              }
         }
         output = input[0];

         // Move input buffer pointer one byte.
         discard;
    };

    condition {             // JIS X 0208
         between 0xa1a1...0xfefe;
    } operation {
         if (codesetnum != 1) {
              if (outputsize <= 4) {
                      error E2BIG;
              }
              // Emit JIS X 0208 sequence, ESC $ B.
              output = 0x1b2442;
              codesetnum = 1;
         } else {
              if (outputsize <= 1) {
                      error E2BIG;
              }
         }
         output = (input[0] & 0x7f);
         output = (input[1] & 0x7f);

         // Move input buffer pointer two bytes.
         discard 2;
    };

    condition {             // JIS X 0201 Kana
```

```
                        between 0x8ea1...0x8edf;
                   } operation {
                        if (codesetnum != 2) {
                             if (outputsize <= 3) {
                                     error E2BIG;
                             }
                             // Emit JIS X 0201 Kana sequence,
                             // ESC ( I.
                             output = 0x1b2849;
                             codesetnum = 2;
                        } else {
                             if (outputsize <= 0) {
                                     error E2BIG;
                             }
                        }
                        output = (input[1] & 127);

                        // Move input buffer pointer two bytes.
                        discard 2;
                   };

                   condition {              // JIS X 0212
                        between 0x8fa1a1...0x8ffefe;
                   } operation {
                        if (codesetnum != 3) {
                             if (outputsize <= 5) {
                                     error E2BIG;
                             }
                             // Emit JIS X 0212 sequence, ESC $ ( D.
                                output = 0x1b242844;
                                codesetnum = 3;
                        } else {
                                if (outputsize <= 1) {
                                        error E2BIG;
                                }
                        }
                        output = (input[1] & 127);
                        output = (input[2] & 127);
                        discard 3;
                   };

                   true    operation {      // error
                        error EILSEQ;
                   };
               };
          }
```

FILES

`/usr/bin/geniconvtbl`
the utility `geniconvtbl`

`/usr/lib/iconv/geniconvtbl/binarytables/*.bt`
conversion binary tables

`/usr/lib/iconv/geniconvtbl/srcs/*`
conversion source files for user reference

SEE ALSO

`cpp`(1), `geniconvtbl`(1), `iconv`(1), `iconv`(3C), `iconv-close`(3C), `iconv-open`(3C), `attributes`(5), `environ`(5)

International Language Environments Guide

NOTES

The maximum length of HEXADECIMAL and DECIMAL digit length is `128`. The maximum length of a variable is `255`. The maximum nest level is `16`.

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*

where

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

The maximum value of the *gid* field is 2137483647. To maximize interoperability and compatibility, administrators are recommended to assign groups using the range of GIDs below 60000 where possible.

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

EXAMPLE 1 Sample of a `group` file.

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`(1), `getgrnam`(3C), `initgroups`(3C), `nsswitch.conf`(4), `unistd`(3HEAD)

System Administration Guide, Volume 1

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

EXAMPLE 1 Example of the `/etc/acct/holidays` file.

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, on consecutive lines. 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.

For a host with more than one IP address, consecutive entries for these addresses may contain the same or differing nicknames. Different nicknames are useful for assigning distinct names to different addresses.

A call to `gethostbyname`(3NSL) returns a `hostent` structure containing the union of all addresses and nicknames from each line containing a matching official name or nickname.

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`(3SOCKET).

This interface supports host names as defined in Internet RFC 952 which states:

A "name" (Net, Host, Gateway, or Domain name) is a text string up to 24 characters drawn from the alphabet (A-Z), digits (0-9), minus sign (–), and period (.). Note that periods are only allowed when they serve to delimit components of "domain style names". (See RFC 921, "Domain Name System Implementation Schedule," for background). No blank or space characters are permitted as part of a name. No distinction is made between upper and lower case. The first character must be an alpha character. The last character must not be a minus sign or period.

Although the interface accepts host names longer than 24 characters for the host portion (exclusive of the domain component), choosing names for hosts that adhere to the 24 character restriction will insure maximum interoperability on the Internet.

A host which serves as a GATEWAY should have “–GATEWAY“ or “–GW” as part of its name. Hosts which do not serve as Internet gateways should not use “–GATEWAY” and “–GW” as part of their names. A host which is a TAC should have “–TAC” as the last part of its host name, if it is a DoD host. Single character names or nicknames are not allowed.

RFC 952 has been modified by RFC 1123 to relax the restriction on the first character being a digit.

EXAMPLES

EXAMPLE 1 Example of a typical line from the `hosts` file.

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

```
192.9.1.20        gaia                           # John Smith
```

SEE ALSO

`in.named`(1M), `gethostbyname`(3NSL), `inet`(3SOCKET), `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`(3SOCKET) . 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`(3SOCKET)) 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. The order of entries is important. If the files contain both 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*]

Hostnames must be the official name of the host, not one of its nicknames.

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.

Search Sequence

To help maintain system security, the `/etc/hosts.equiv` file is not checked when access is being attempted for super-user. If the user attempting access is not the super-user, `/etc/hosts.equiv` is searched for lines of the form described above. Checks are made for lines in this file in the following order:

1. `+`
2. `+@` *netgroup*
3. `-@` *netgroup*
4. `-` *hostname*
5. *hostname*

The user is granted access if a positive match occurrs. Negative entries apply only to `/etc/hosts.equiv` and may be overridden by subsequent `.rhosts` entries.

If no positive match occurred, the `.rhosts` file is then searched if the user attempting access maintains such a file. This file is searched whether or not the user attempting access is the super-user. As a security feature, the `.rhosts` file must be owned by the user who is attempting access. Checks are made for lines in `.rhosts` in the following order:

1. `+`
2. `+@` *netgroup*
3. `-@` *netgroup*
4. `-` *hostname*
5. *hostname*

FILES

`/etc/hosts.equiv`	system trusted hosts and users
`~/.rhosts`	user's trusted hosts and users

SEE ALSO

`rcp`(1) , `rlogin`(1) , `rsh`(1) , `rcmd`(3SOCKET) , `hosts`(4) , `netgroup`(4) , `passwd`(4)

WARNINGS

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. For example, because of the search sequence, an `/etc/hosts.equiv` file consisting of the entries

```
+
-hostxxx
```

will not deny access to "hostxxx".

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
: A recognized protocol listed in the file `/etc/inet/protocols`. For servers capable of supporting TCP and UDP over IPv6, the following protocol types are also recognized:

`tcp6`

`udp6`

`tcp6` and `udp6` are not official protocols; accordingly, they are not listed in the `/etc/inet/protocols` file.

Here the `inetd` program uses an `AF_INET6` type socket endpoint. These servers can also handle incoming IPv4 client requests in addition to IPv6 client requests.

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
: This field has values `wait` or `nowait`. This entry specifies whether the server that is invoked by `inetd` will take over the listening socket associated with the service, and whether once launched, `inetd` will `wait` for that server to exit, if ever, before it resumes listening for new service requests. The *wait-status* for datagram servers must be set to `wait`, as they are always invoked with the orginal datagram socket that will participate in delivering the service bound to the specified service. They do not have separate "listening" and "accepting" sockets. Accordingly, do not configure UDP services as nowait. This causes a race condition by which 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 severely compromised. Connection-oriented services such as TCP stream services can be designed to be either `wait` or `nowait` status.

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 20 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

inet_type – default Internet protocol type

SYNOPSIS

/etc/default/inet_type

DESCRIPTION

The inet_type file defines the default IP protocol to use. Currently this file is only used by the ifconfig(1M) and netstat(1M) commands.

The inet_type file can contain a number of <variable>=<value> lines. Currently, the only variable defined is DEFAULT_IP, which can be assigned a value of IP_VERSION4, IP_VERSION6, or BOTH.

The output displayed by the ifconfig and netstat commands can be controlled by the value of DEFAULT_IP set in inet_type file. By default, both commands display the IPv4 and IPv6 information available on the system. The user can choose to suppress display of IPv6 information by setting the value of DEFAULT_IP. The following shows the possible values for DEFAULT_IP and the resulting ifconfig and netstat output that will be displayed:

IP_VERSION4
: Displays only IPv4 related information. The output displayed is backward compatible with older versions of the ifconfig(1M) and netstat(1M) commands.

IP_VERSION6
: Displays both IPv4 and IPv6 related information for ifconfig and netstat.

BOTH
: Displays both IPv4 and IPv6 related information for ifconfig and netstat.

The command-line options to the ifconfig and netstat commands override the effect of DEFAULT_IP as set in the inet_type file. For example, even if the value of DEFAULT_IP is IP_VERSION4, the command

```
example% ifconfig -a6
```

will display all IPv6 interfaces.

EXAMPLES

EXAMPLE 1 Suppressing IPv6 Related Output

This is what the inet_type file must contain if you want to suppress IPv6 related output:

```
DEFAULT_IP=IP_VERSION4
```

SEE ALSO

ifconfig(1M), netstat(1M)

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

EXAMPLE 1 Example of /sbin/rc2.

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 'start', and will terminate the daemons if given the 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

ipnodes – local database associating names of nodes with IP addresses

SYNOPSIS

/etc/inet/ipnodes

DESCRIPTION

The ipnodes file is a local database that associates the names of nodes with their Internet Protocol (IP) addresses. IP addresses can be either an IPv4 or an IPv6 address. The ipnodes file can be used in conjunction with, or instead of, other ipnodes databases, including the Domain Name System (DNS), the NIS ipnodes map, and the NIS+ ipnodes table. Programs use library interfaces to access information in the ipnodes file.

The ipnodes file has one entry for each IP address of each node. If a node has more than one IP address, it will have one entry for each, on consecutive lines. The format of each line is:

```
IP-address official-node-name nicknames...
```

Items are separated by any number of SPACE and / or TAB characters. The first item on a line is the node's IP address. The second entry is the node's official name. Subsequent entries on the same line are alternative names for the same machine, or "nicknames." Nicknames are optional.

For a node with more than one IP address, consecutive entries for these addresses may contain the same or differing nicknames. Different nicknames are useful for assigning distinct names to different addresses.

A call to getipnodebyname(3SOCKET) returns a hostent structure containing the union of all addresses and nicknames from each line containing a matching official name or nickname.

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 one of two ways:

- The conventional "decimal dot" notation and interpreted using the inet_addr routine from the Internet address manipulation library, inet(3SOCKET).
- The IP Version 6 protocol [IPV6], defined in *RFC 1884* and interpreted using the inet_pton( ) routine from the Internet address manipulation library. See inet(3SOCKET).

These interfaces supports node names as defined in *Internet RFC 952* which states:

A "name" (Net, Host, Gateway, or Domain name) is a text string up to 24 characters drawn from the alphabet (A-Z), digits (0-9), minus sign (-), and period (.). Note that periods are only allowed when they serve to delimit components of "domain style names". (See *RFC 921, "Domain Name System Implementation Schedule,"* for background). No blank or space characters are permitted as part of a name. No distinction is made between upper and lower case. The first character must be an alpha character. The last character must not be a minus sign or period.

Although the interface accepts node names longer than 24 characters for the node portion (exclusive of the domain component), choosing names for nodes that adhere to the 24 character restriction will insure maximum interoperability on the Internet.

A node which serves as a GATEWAY should have "`–GATEWAY`" or "`–GW`" as part of its name. Nodes which do not serve as Internet gateways should not use "`–GATEWAY`" and "`–GW`" as part of their names. A node that is a TAC should have "`–TAC`" as the last part of its node name, if it is a DoD node. Single character names or nicknames are not allowed.

RFC 952 has been modified by *RFC 1123* to relax the restriction on the first character being a digit.

EXAMPLES

EXAMPLE 1 A Typical Line from the `ipnodes` File

The following is a typical line from the `ipnodes` file:

```
2::56:a00:20ff:fe7b:b667        foo             # John Smith
```

SEE ALSO

`in.named`(1M), `getipnodebyname`(3SOCKET), `inet`(3SOCKET), `nsswitch.conf`(4), `resolv.conf`(4), `hosts`(4)

Braden, B., editor, *RFC 1123, Requirements for Internet Hosts – Application and Support*, Network Working Group, October, 1989.

Harrenstien, K., Stahl, M., and Feinler, E., *RFC 952, DOD INTERNET HOST TABLE SPECIFICATION*, Network Working Group, October 1985.

Hinden, R., and Deering, S., editors, *RFC 1884, IP Version 6 Addressing Architecture*, Network Working Group, December, 1995.

Postel, Jon, *RFC 921, Domain Name System Implementation Schedule — Revised*, Network Working Group, October 1984.

NOTES

IPv4 addresses can be defined in the `ipnodes` file or in the `hosts` file. See `hosts`(4). The `ipnodes` file will be searched for IPv4 addresses when using the `getipnodebyname`(3SOCKET) API. If no matching IPv4 addresses are found in the `ipnodes` file, then the `hosts` file will be searched. To prevent delays in name resolution and to keep `/etc/inet/ipnodes` and `/etc/inet/hosts` synchronized, IPv4 addresses defined in the `hosts` file should be copied to the `ipnodes` file.

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

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

EXAMPLE 1 Example of setting multiple keystations.

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
```

CODE EXAMPLE 1 Exchange DELETE and BACKSPACE keys

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.

CODE EXAMPLE 2 Disable CAPS LOCK key

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

```
key 119 all nop
```

CODE EXAMPLE 3 Standard translation tables for the U.S. Type 4 keyboard

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

krb5.conf – Kerberos configuration file

SYNOPSIS

/etc/krb5/krb5.conf

DESCRIPTION

The `krb5.conf` file contains Kerberos configuration information, including the locations of KDCs and administration daemons for the Kerberos realms of interest, defaults for the current realm and for Kerberos applications, and mappings of host names onto Kerberos realms. This file must reside on all Kerberos clients.

The format of the `krb5.conf` consists of sections headings in square brackets. Each section may contain zero or more configuration variables (called *relations*), of the form:

relation= *relation-value*

or

relation-subsection = {
relation= *relation-value*
relation= *relation-value*

}

The `krb5.conf` file may contain any or all of the following seven sections:

`libdefaults` Contains default values used by the Kerberos V5 library.

`appdefaults` Contains subsections for Kerberos V5 applications, where *relation-subsection* is the name of an application. Each subsection describes application-specific defaults.

`realms` Contains subsections for Kerberos realms, where *relation-subsection* is the name of a realm. Each subsection contains relations that define the properties for that particular realm.

`domain_realm` Contains relations which map domain names and subdomains onto Kerberos realm names. This is used by programs to determine what realm a host should be in, given its fully qualified domain name.

`logging` Contains relations which determine how Kerberos programs are to perform logging.

`capaths` — Contains the authentication paths used with direct (nonhierarchical) cross-realm authentication. Entries in this section are used by the client to determine the intermediate realms which may be used in cross-realm authentication. It is also used by the end-service when checking the transited field for trusted intermediate realms.

`kdc` — For a KDC, may contain the location of the `kdc.conf` file.

[libdefaults]

The `[libdefaults]` section may contain any of the following relations:

`default_realm` — Identifies the default Kerberos realm for the client. Set its value to your Kerberos realm.

`default_tgs_enctypes` — Identifies the supported list of session key encryption types that should be returned by the KDC. The list may be delimited with commas or whitespace. The supported encryption types are `des-cbc-crc` and `des-cbc-md5`.

`default_tkt_enctypes` — Identifies the supported list of session key encryption types that should be requested by the client. The format is the same as for `default_tkt_enctypes`. The supported encryption types are `des-cbc-crc` and `des-cbc-md5`.

`clockskew` — Sets the maximum allowable amount of clock skew in seconds that the library will tolerate before assuming that a Kerberos message is invalid. The default value is 300 seconds, or five minutes.

[appdefaults]

This section contains subsections for Kerberos V5 applications, where *relation-subsection* is the name of an application. Each subsection contains relations that define the default behaviors for that application.

```
gkadmin = {
      help_url = http://localhost:8888/ab2/coll.384.1/SEAM
}
```

The following application defaults can be set to `true` or `false`:

```
kinit
   forwardable
   proxiable
   renewable
```

```
    max_life = delta_time
    max_renewable_life = delta_time

 (See kinit(1) for the valid time duration formats
 you can specify for delta_time.)
```

In the following example, `kinit` will get forwardable tickets by default, and `telnet` has three default behaviors specified:

```
 [appdefaults]
    kinit = {
       forwardable = true
    }

    telnet = {
       forward = true
       encrypt = true
       autologin = true
    }
```

The application defaults specified here are overridden by those specified in the `[realms]` section.

[realms]

This section contains subsections for Kerberos realms, where *relation-subsection* is the name of a realm. Each subsection contains relations that define the properties for that particular realm. The following relations may be specified in each `[realms]` subsection:

`kdc` — The name of a host running a KDC for that realm. An optional port number (separated from the hostname by a colon) may be included.

`admin_server` — Identifies the host where the Kerberos administration daemon (`kadmind`) is running. Typically, this is the master KDC.

application defaults — Application defaults that are specific to a particular realm may be specified within a `[realms]` subsection. Realm-specific application defaults override the global defaults specified in the `[appdefaults]` section.

[domain_realm]

This section provides a translation from a domain name or hostname to a Kerberos realm name. The *relation* can be a host name, or a domain name, where domain names are indicated by a period (‘`.`’) prefix. *relation-value* is the Kerberos realm name for that particular host or domain. Host names and domain names should be in lower case.

If no translation entry applies, the host’s realm is considered to be the hostname’s domain portion converted to upper case. For example, the following [domain_realm] section maps crash.mit.edu into the TEST.ATHENA.MIT.EDU realm:

```
[domain_realm]
   .mit.edu = ATHENA.MIT.EDU
   mit.edu = ATHENA.MIT.EDU
   crash.mit.edu = TEST.ATHENA.MIT.EDU
   .fubar.org = FUBAR.ORG
   fubar.org = FUBAR.ORG
```

All other hosts in the mit.edu domain will map by default to the ATHENA.MIT.EDU realm, and all hosts in the fubar.org domain will map by default into the FUBAR.ORG realm. Note the entries for the hosts mit.edu and fubar.org. Without these entries, these hosts would be mapped into the Kerberos realms EDU and ORG, respectively.

[logging]

This section indicates how Kerberos programs are to perform logging. There are two types of relations for this section: relations to specify how to log and a relation to specify how to rotate kdc log files.

The following relations may be defined to specify how to log. The same relation can be repeated if you want to assign it multiple logging methods.

`admin_server` Specifies how to log the Kerberos administration daemon (kadmind). The default is FILE:/var/krb5/kadmin.log.

`default` Specifies how to perform logging in the absence of explicit specifications otherwise.

`kdc` Specifies how the KDC is to perform its logging. The default is FILE:/var/krb5/kdc.log.

The admin_server, default, and kdc relations may have the following values:

FILE:*filename*

or

FILE=*filename* This value causes the entity's logging messages to go to the specified file. If the '=' form is used, the file is overwritten. If the ':' form is used, the file is appended to.

STDERR This value causes the entity's logging messages to go to its standard error stream.

CONSOLE	This value causes the entity's logging messages to go to the console, if the system supports it.
DEVICE=*devicename*	This causes the entity's logging messages to go to the specified device.
SYSLOG[:*severity*[:*facility*]]	This causes the entity's logging messages to go to the system log.

The *severity* argument specifies the default severity of system log messages. This may be any of the following severities supported by the syslog(3C) call, minus the LOG_ prefix: LOG_EMERG, LOG_ALERT, LOG_CRIT, LOG_ERR, LOG_WARNING, LOG_NOTICE, LOG_INFO, and LOG_DEBUG. For example, a value of CRIT would specify LOG_CRIT severity.

The *facility* argument specifies the facility under which the messages are logged. This may be any of the following facilities supported by the syslog(3C) call minus the LOG_ prefix: LOG_KERN, LOG_USER, LOG_MAIL, LOG_DAEMON, LOG_AUTH, LOG_LPR, LOG_NEWS, LOG_UUCP, LOG_CRON, and LOG_LOCAL0 through LOG_LOCAL7.

If no severity is specified, the default is ERR. If no facility is specified, the default is AUTH.

The following relation may be defined to specify how to rotate kdc log files if the FILE: value is being used to log:

kdc_rotate	A relation subsection that enables kdc logging to be rotated to multiple files based on a time interval. This can be used to avoid logging to one file, which may grow too large and bring the KDC to a halt.

The time interval for the rotation is specified by the period relation. The number of log files to be rotated is specified by the versions relation. Both the period and versions (described below) should be included in this subsection. And, this subsection applies only if the kdc relation has a FILE: value.

The following relations may be specified for the kdc_rotate relation subsection:

period=*delta_time*	Specifies the time interval before a new log file is created. See the Time Formats section in kinit(1) for the valid time duration formats you can specify for *delta_time*. If period is not specified or set to "never", no rotation will occur.

Specifying a time interval does not mean that the log files will be rotated at the time interval based on real time. This is because the time interval is checked at each attempt to write a record to the log, or when logging is actually occurring. Therefore, rotation occurs only when logging has actually occurred for the specified time interval.

`versions=`*number* Specifies how many previous versions will be saved before the rotation begins. A number will be appended to the log file, starting with 0 and ending with (*number* - 1). For example, if `versions` is set to `2`, up to three logging files will be created (*filename*, *filename*.0, and *filename*.1) before the first one is overwritten to begin the rotation.

Notice that if `versions` is not specified or set to `0`, only one log file will be created, but it will be overwritten whenever the time interval is met.

In the following example, the logging messages from the Kerberos administration daemon will go to the console. The logging messages from the KDC will be appended to the `/var/krb5/kdc.log`, which will be rotated between twenty-one log files with a specified time interval of a day.

```
[logging]
   admin_server = CONSOLE
   kdc = FILE:/export/logging/kadmin.log
   kdc_rotate = {
      period = 1d
      versions = 20
   }
```

[capaths] In order to perform direct (non-hierarchical) cross-realm authentication, a database is needed to construct the authentication paths between the realms. This section defines that database.

A client will use this section to find the authentication path between its realm and the realm of the server. The server will use this section to verify the authentication path used by the client, by checking the transited field of the received ticket.

There is a subsection for each participating realm, and each subsection has relations named for each of the realms. The *relation-value* is an intermediate realm which may participate in the cross-realm authentication. The relations may be repeated if there is more than one intermediate realm. A value of '.' means that the two realms share keys directly, and no intermediate realms should be allowed to participate.

There are n**2 possible entries in this table, but only those entries which will be needed on the client or the server need to be present. The client needs a subsection named for its local realm, with relations named for all the realms of servers it will need to authenticate with. A server needs a subsection named for each realm of the clients it will serve.

For example, ANL.GOV, PNL.GOV, and NERSC.GOV all wish to use the ES.NET realm as an intermediate realm. ANL has a sub realm of TEST.ANL.GOV, which will authenticate with NERSC.GOV but not PNL.GOV. The [capath] section for ANL.GOV systems would look like this:

```
[capaths]
   ANL.GOV = {
       TEST.ANL.GOV = .
       PNL.GOV = ES.NET
       NERSC.GOV = ES.NET
       ES.NET = .
   }

   TEST.ANL.GOV = {
       ANL.GOV = .
   }

   PNL.GOV = {
       ANL.GOV = ES.NET
   }

   NERSC.GOV = {
      ANL.GOV = ES.NET
   }

   ES.NET = {
      ANL.GOV = .
   }
```

The [capath] section of the configuration file used on NERSC.GOV systems would look like this:

```
[capaths]
   NERSC.GOV = {
      ANL.GOV = ES.NET
      TEST.ANL.GOV = ES.NET
      TEST.ANL.GOV = ANL.GOV
      PNL.GOV = ES.NET
      ES.NET = .
   }

   ANL.GOV = {
      NERSC.GOV = ES.NET
   }

   PNL.GOV = {
      NERSC.GOV = ES.NET
   }
```

```
    ES.NET = {
       NERSC.GOV = .
    }

    TEST.ANL.GOV = {
       NERSC.GOV = ANL.GOV
       NERSC.GOV = ES.NET
    }
```

In the above examples, the ordering is not important, except when the same relation is used more than once. The client will use this to determine the path. (It is not important to the server, since the transited field is not sorted.)

EXAMPLES

EXAMPLE 1 Sample file

Here is an example of a generic `krb5.conf` file:

```
 [libdefaults]
    ticket_lifetime = 600
    default_realm = ATHENA.MIT.EDU
    default_tkt_enctypes = des-cbc-crc
    default_tgs_enctypes = des-cbc-crc

 [realms]
    ATHENA.MIT.EDU = {
       kdc = kerberos.mit.edu
       kdc = kerberos-1.mit.edu
       kdc = kerberos-2.mit.edu
       admin_server = kerberos.mit.edu
       default_domain = mit.edu
    }

    FUBAR.ORG = {
       kdc = kerberos.fubar.org
       kdc = kerberos-1.fubar.org
       admin_server = kerberos.fubar.org
   }

 [domain_realm]
    .mit.edu = ATHENA.MIT.EDU
    mit.edu = ATHENA.MIT.EDU
```

FILES

`/var/krb5/kdc.log` KDC logging file

SEE ALSO

`kinit`(1), `syslog`(3C), SEAM(5)

NOTES

If the `krb5.conf` file is not formatted properly, the `telnet` command will fail. However, the `dtlogin` and `login` commands will still succeed, even if the `krb5.conf` file is specified as required for the commands. If this occurs, the following error message will be displayed:

```
Error initializing krb5: Improper format of
```

To bypass any other problems that may occur, you should fix the file as soon as possible.

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`(3KRB)

BUGS

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

NAME

ldapfilter.conf – configuration file for LDAP filtering routines

SYNOPSIS

`/etc/opt/SUNWconn/ldap/current/ldapfilter.conf`

DESCRIPTION

The `ldapfilter.conf` file contains information used by the LDAP filtering routines.

Blank lines and lines that begin with a hash character ('#') are treated as comments and ignored. The configuration information consists of lines that contain one to five tokens. Tokens are separated by white space, and double quotes can be used to include white space inside a token.

The file consists of a sequence of one or more filter sets. A filter set begins with a line containing a single token called a *tag.*

The filter set consists of a sequence of one or more filter lists. The first line in a filter list must contain four or five tokens: the *value pattern,* the *delimiter list,* a *filter template,* a *match description,* and an optional *search scope.* The *value pattern* is a regular expression that is matched against the `value` passed to the LDAP library call to select the filter list.

The *delimiter list* is a list of the characters (in the form of a single string) that can be used to break the `value` into distinct words.

The *filter template* is used to construct an LDAP filter (see description below)

The *match description* is returned to the caller along with a filter as a piece of text that can be used to describe the sort of LDAP search that took place. It should correctly compete both of the following phrases: "One *match description* match was found for..." and "Three *match description* matches were found for...."

The *search scope* is optional, and should be one of "base", "onelevel", or "subtree". If *search scope* is not provided, the default is "subtree".

The remaining lines of the filter list should contain two or three tokens, a *filter template,* a *match description* and an optional *search scope* .

The *filter template* is similar in concept to a `printf`(3C) style format string. Everything is taken literally except for the character sequences:

`%v`	Substitute the entire `value` string in place of the `%v`.
`%v$`	Substitute the last word in this field.
`%v`*N*	Substitute word *N* in this field (where *N* is a single digit 1-9). Words are numbered from left to right within the value starting at 1.
`%v`*M-N*	Substitute the indicated sequence of words where *M* and *N* are both single digits 1-9.

`%vN-` Substitute word *N* through the last word in `value` where *N* is again a single digit 1-9.

EXAMPLES

EXAMPLE 1 The following ldap filter configuration file contains two filter sets, `example1` and `example2 onelevel`, each of which contains four filter lists.

```
# ldap filter file
#
example1
"="                  " "     "%v"                       "arbitrary filter"
"[0-9][0-9-]*"      " "     "(telephoneNumber=*%v)" "phone number"

"@"                  " "     "(mail=%v)"                "email address"

"^.[. _].*"          ". _"   "(cn=%v1* %v2-)"           "first initial"

".*[. _].$"          ". _"   "(cn=%v1-*)"               "last initial"

"[. _]"              ". _"   "(|(sn=%v1-)(cn=%v1-))"        "exact"
                             "(|(sn~=%v1-)(cn~=%v1-))"      "approximate"

".*"                 ". "    "(|(cn=%v1)(sn=%v1)(uid=%v1))" "exact"
                             "(|(cn~=%v1)(sn~=%v1))"        "approximate"

"example2 onelevel"
"^..$"  " "   "(|(o=%v)(c=%v)(l=%v)(co=%v))"       "exact" "onelevel"
              "(|(o~=%v)(c~=%v)(l~=%v)(co~=%v))"   "approximate"
"onelevel"

" "     " "   "(|(o=%v)(l=%v)(co=%v)"       "exact"        "onelevel"
              "(|(o~=%v)(l~=%v)(co~=%v)"    "approximate"  "onelevel"

"."    " "   "(associatedDomain=%v)"       "exact"        "onelevel"

".*"    " "   "(|(o=%v)(l=%v)(co=%v)"       "exact"        "onelevel"
              "(|(o~=%v)(l~=%v)(co~=%v)"    "approximate"  "onelevel"
```

ATTRIBUTES

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

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

SEE ALSO

`ldap_getfilter`(3LDAP), `ldap_ufn`(3LDAP), `attributes`(5)

NAME

ldapsearchprefs.conf – configuration file for LDAP search preference routines

SYNOPSIS

`/etc/opt/SUNWconn/ldap/current/ldapsearchprefs.conf`

DESCRIPTION

The `ldapsearchprefs.conf` file contains information used by LDAP when searching the directory. Blank lines and lines that start with a hash ('#') character are treated as comments and ignored. Non-comment lines contain one or more tokens. Tokens are separated by white space, and double quotes can be used to include white space inside a token.

Search preferences are typically used by LDAP-based client programs to specify what a user may search for, which attributes are searched, and which options are available to the user.

The first non-commment line specifies the version of the template information and must contain the token `Version` followed by an integer version number. For example:

```
Version 1
```

The current version is *1*, so the above example is always the correct opening line.

The remainder of the file consists of one or more search preference configurations. The first line of a search preference is a human-readable name for the type of object being searched for, for example `People` or `Organizations`. This name is stored in the *so_objtypeprompt* member of the `ldap_searchobj` structure (see `ldap_searchprefs`(3LDAP)). For example,

```
People
```

specifies a label for a search preference designed to find X.500 entries for people.

The next line specifies a list of options for this search object. The only option currently allowed is "internal" which means that this search object should not be presented directly to a user. Options are placed in the *so_options* member of the *ldap_searchobj* structure and can be tested using the `LDAP_IS_SEARCHOBJ_OPTION_SET()` macro. Use "" if no special options are required.

The next line specifes a label to use for "Fewer Choices" searches. "Fewer Choices" searches are those where the user's input is fed to the ldap_filter routines to determine an appropriate filter to use. This contrasts with explicitly-constructed LDAP filters, or "More Choices" searches, where the user can explicitly construct an LDAP filter.

For example:

```
"Search For:"
```

can be used by LDAP client programs to label the field into which the user can type a "Fewer Choices" search.

The next line specifies an LDAP filter prefix to append to all "More Choices" searched. This is typically used to limit the types of entries returned to those containing a specific object class. For example:

```
"(&(objectClass=person)"
```

would cause only entries containing the object class *person* to be returned by a search. Note that parentheses may be unbalanced here, since this is a filter prefix, not an entire filter.

The next line is an LDAP filter tag which specifies the set of LDAP filters to be applied for "Fewer Choices" searching. The line

```
"x500-People"
```

would tell the client program to use the set of LDAP filters from the ldap filter configuration file tagged "x500-People".

The next line specifies an LDAP attribute to retrieve to help the user choose when several entries match the search terms specified. For example:

```
"title"
```

specifies that if more than one entry matches the search criteria, the client program should retrieve the `title` attribute that and present that to the user to allow them to select the appropriate entry. The next line specifies a label for the above attribute, for example,

```
"Title:"
```

Note that the values defined so far in the file are defaults, and are intended to be overridden by the specific search options that follow.

The next line specifies the scope of the LDAP search to be performed. Acceptable values are subtree, onelevel, and base.

The next section is a list of "More Choices" search options, terminated by a line containing only the string `END`. For example:

```
"Common Name" cn 11111 "" ""
"Surname" sn 11111 "" ""
"Business Phone" "telephoneNumber" 11101 "" ""
END
```

Each line represents one method of searching. In this example, there are three ways of searching - by Common Name, by Surname, and by Business Phone number. The first field is the text which should be displayed to user. The second field is the attribute which will be searched. The third field is a bitmap which specifies which of the match types are permitted for this search type. A "1" value in a given bit position indicates that a particular match type is valid, and a "0" indicates that is it not valid. The fourth and fifth fields are, respectively, the select attribute name and on-screen name for the selected attribute. These values are intended to override the defaults defined above. If no specific values are specified, the client software uses the default values above.

The next section is a list of search match options, terminated by a a line containing only the string END. Example:

```
"exactly matches" "(%a=%v))"
"approximately matches" "(%a~=%v))"
"starts with" "(%a=%v*))"
"ends with" "(%a=*%v))"
"contains" "(%a=*%v*))"
END
```

In this example, there are five ways of refining the search. For each method, there is an LDAP filter suffix which is appended to the ldap filter.

EXAMPLES

EXAMPLE 1 The following example illustrates one possible configuration of search preferences for "people".

```
# Version number
Version 1
# Name for this search object
People
# Label to place before text box user types in
"Search For:"
# Filter prefix to append to all "More Choices" searches
"(&(objectClass=person)"
# Tag to use for "Fewer Choices" searches - from ldapfilter.conf file
"x500-People"
# If a search results in > 1 match, retrieve this attribute to help
# user distinguish between the entries...
multilineDescription
# ...and label it with this string:
"Description"
# Search scope to use when searching
subtree
# Follows a list of "More Choices" search options.  Format is:
# Label, attribute, select-bitmap, extra attr display name, extra attr ldap name
# If last two are null, "Fewer Choices" name/attributes used
"Common Name"                   cn                 11111  ""  ""
"Surname"                       sn                 11111  ""  ""
"Business Phone"                "telephoneNumber"  11101  ""  ""
"E-Mail Address"                "mail"             11111  ""  ""
"Uniqname"                      "uid"              11111  ""  ""
END
# Match types
"exactly matches"               "(%a=%v))"
"approximately matches"         "(%a~=%v))"
"starts with"                   "(%a=%v*))"
"ends with"                     "(%a=*%v))"
"contains"                      "(%a=*%v*))"
END
```

In this example, the user may search for People. For "fewer choices" searching, the tag for the `ldapfilter.conf`(4) file is "x500-People".

ATTRIBUTES

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

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

SEE ALSO `ldap_searchprefs`(3LDAP) `attributes`(5)

NAME

ldaptemplates.conf – configuration file for LDAP display template routines

SYNOPSIS

`/etc/opt/SUNWconn/ldap/current/ldaptemplates.conf`

DESCRIPTION

The `ldaptemplates.conf` file contains information used by the LDAP display routines.

Blank lines and lines that start with a hash character ('#') are treated as comments and ignored. Non-comment lines contain one or more tokens. Tokens are separated by white space, and double quotes can be used to include white space inside a token.

The first non-commment line specifies the version of the template information and must contain the token `Version` followed by an integer version number. For example,

```
Version 1
```

The current version is *1*, so the above example is always the correct first line.

The remainder of the file consists of one or more display templates. The first two lines of the display template each contain a single token that specifies singular and plural names for the template in a user-friendly format. For example,

```
"Person"
"People"
```

specifies appropriate names for a template designed to display person information.

The next line specifies the name of the icon or similar element that is associated with this template. For example,

```
"person icon"
```

The next line is a blank-separated list of template options. "" can be used if no options are desired. Available options are: `addable` (it is appropriate to allow entries of this type to be added), `modrdn` (it is appropriate to offer the `modify rdn` operation), `altview` (this template is an alternate view of another template). For example,

```
"addable" "modrdn"
```

The next portion of the template is a list of X.500 object classes that is used to determine whether the template should be used to display a given entry. The object class information consists of one or more lines, followed by a terminating line that contains the single token `END`. Each line contains one or more object class names, all of which must be present in a directory entry. Multiple lines can be used to associate more than one set of object classes with a given template. For example,

```
emailPerson
orgPerson
END
```

means that the template is appropriate for display of `emailPerson` entries or `orgPerson` entries.

The next line after the object class list is the name of the attribute to authenticate as to make changes (use "" if it is appropriate to authenticate as the entry itself). For example,

```
"owner"
```

The next line is the default attribute to use when naming a new entry, for example,

```
"cn"
```

The next line is the distinguished name of the default location under which new entries are created. For example,

```
"o=XYZ, c=US"
```

The next section is a list of rules used to assign default values to new entries. The list should be terminated with a line that contains the single token `END`. Each line in this section should either begin with the token `constant` and be followed by the name of the attribute and a constant value to assign, or the line should begin with `addersdn` followed by the name of an attribute whose value will be the DN of the person who has authenticated to add the entry. For example,

```
constant associatedDomain XYZ.us
addersdn seeAlso
END
```

The last portion of the template is a list of items to display. It consists of one or more lines, followed by a terminating line that contains the single token `END`. Each line is must begin with the token `samerow` or the token `item`

It is assumed that each item appears on a row by itself unless it was preceded by a `samerow` line (in which case it should be displayed on the same line as the previous item, if possible). Lines that begin with `samerow` should not have any other tokens on them.

Lines that begin with `item` must have at least three more tokens on them: an item type, a label, and an attribute name. Any extra tokens are taken as extra arguments.

The item type token must be one of the following strings:

`cis`	case-ignore string attributes
`mls`	multiline string attributes
`mail`	RFC-822 conformant mail address attributes
`dn`	distinguished name pointer attributes
`bool`	Boolean attributes
`jpeg`	JPEG photo attributes
`jpegbtn`	a button that will retrieve and show a JPEG photo attribute
`fax`	FAX T.4 format image attributes
`faxbtn`	a button that will retrieve and show a FAX photo attribute
`audiobtn`	audio attributes
`time`	UTC time attributes
`date`	UTC time attributes where only the date portion should be shown
`url`	labeled Uniform Resource Locator attributes
`searchact`	define an action that will do a directory search for other entries
`linkact`	define an action which is a link to another display template
`protected`	for an encrypted attribute, with values displayed as asterisks

An example of an item line for the drink attribute (displayed with label "Work Phone"):

```
item cis "Work Phone" telephoneNumber
```

EXAMPLES

EXAMPLE 1 The following template configuration file contains a templates for display of people entries.

```
#
 # LDAP display templates
 #
 # Version must be 1 for now
 #
 Version 1
 #
 # Person template
 "Person"
 "People"

 # name of the icon that is associated with this template
 "person icon"
```

```
# blank-separated list of template options ("" for none)
"addable"

#
# objectclass list
person
END

#
# name of attribute to authenticate as ("" means auth as this entry)
""

#
# default attribute name to use when forming RDN of a new entry
#
"cn"

#
# default location when adding new entries (DN; "" means no default)
"o=XYZ, c=US"

#
# rules used to define default values for new entries
END

#
# list of items for display
item jpegbtn "View Photo"  jpegPhoto "Next Photo"
item audiobtn "Play Sound"  audio
item cis "Also Known As"  cn
item cis "Title"   title
item mls "Work Address"  postalAddress
item cis "Work Phone"  telephoneNumber
item cis "Fax Number"  facsimileTelephoneNumber
item mls "Home Address"  homePostalAddress
item cis "Home Phone"  homePhone
item cis "User ID"  uid
item mail "E-Mail Address" mail
item cis "Description"  description
item dn  "See Also"  seeAlso
END
```

ATTRIBUTES

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

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

SEE ALSO

`ldap_disptmpl`(3LDAP) `ldap_entry2text`(3LDAP) `attributes`(5)

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_MAX32	1048320	/* max length of arguments to exec 32-bit program */
_ARG_MAX64	2096640	/* max length of arguments to exec 64-bit program */
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	2147483647L	/* max value of a long int if _ILP32 defined */
	9223372036854775807L	/* max value of a long int if _LP64 defined */
LONG_MIN	(-2147483647-1L)	/* min value of a long int if _ILP32 defined */

	(-9223372036854775807L-1L)	/* min value of a long int if _LP64 defined */
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	999999	/* 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	2147483647	/* max value for a user or group ID */
UINT_MAX	4294967295	/* max value of an "unsigned int" */
ULONG_MAX	4294967295UL	/* max value of an "unsigned long int" if _ILP32 defined */
	18446744073709551615UL	/* max value of an "unsigned long int" if _LP64 defined */
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-conforming application (see `standards`(5)). 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 */

SEE ALSO

standards(5)

NAME

llc2 – LLC2 Configuration file

SYNOPSIS

/etc/llc2/default/llc2.*

DESCRIPTION

The *llc2* files contain information needed by LLC2 to establish the appropriate links to the underlying MAC layer drivers as well as the parameters necessary to configure the LLC (Logical Link Control) Class II Station Component structures for that link.

The comments are made up of one or more lines starting with the "#" character in column 1.

The main section consists of keyword/value pairs of the form *keyword=value*, used to initialize the particular adapter.

A sample of the *llc2* is presented below:

```
devicename=/dev/dnet
deviceinstance=1
llc2_on=1         # LLC2: On/Off on this device
deviceloopback=1
timeinterval=0    # LLC2: Timer Multiplier
acktimer=2        # LLC2: Ack Timer
rsptimer=2        # LLC2: Response Timer
polltimer=4       # LLC2: Poll Timer
rejecttimer=6     # LLC2: Reject Timer
rembusytimer=8    # LLC2: Remote Busy Timer
inacttimer=30     # LLC2: Inactivity Timer
maxretry=6        # LLC2: Maximum Retry Value
xmitwindowsz=14   # LLC2: Transmit Window Size
rcvwindowsz=14    # LLC2: Receive Window Size
```

MAC specific Parameters

The *llc2.ppa* file contains 4 parameters directly related to the underlying MAC-level driver. These are the name of the physical device, the instance of the device, whether LLC2 can be used with this device, and whether the device is capable of looping back data addressed to the node's unique MAC address, broadcast address, or multicast addresses.

Setting the `llc2_on` parameter to `1` means that LLC2 can be used with this device; setting it to `0` means otherwise. Setting the loopback parameter to `1` means that the LLC2 module will loop back data addressed to this node's unique MAC address or to a broadcast/multicast address.

The most likely use is for a media that cannot receive its own transmissions (for example, ethernet) or when the MAC-level driver intentionally does not loop back data addressed to the local node under the assumption that the upper layers have already done so.

Host-Based LLC2 Parameters

The LLC2 contains ten parameters in the configuration file (`/etc/llc2/default/llc2.`*ppa*) that apply to configurations using the

Host-Based LLC2 component for connection-oriented operation over an Ethernet, Token Ring, or FDDI media.

The ten parameters break down into the following four groups:

- Six parameters deal with timer settings for managing the flow of LLC elements of procedure (PDUs) on a data link connection.
- One parameter is the multiplier that is used to determine the period of the interval timer for the station. A value of `1` means that each tick count represents 100 milliseconds; `5` means each tick count is 500 milliseconds. Should the parameter be omitted, the default value is `5`, except for Token Ring links which use a default of `1`.
- One parameter indicates how many times an operation should be retried on a data link connection.
- Two parameters are for controlling the number of unacknowledged I PDUs to send or receive on a data link connection.

Additional information on these parameters can be found in ISO 8802-2 : 1989, Section 7.8.

The following table of `Logical Link Control Parameters` provides the LLC configuration parameter names, default values, and ranges.

Parameter	Description	Default	Range
`timeinterval`	The timer ticks in 100 ms intervals. This parameter is used to scale the following 5 timer parameters.	5, except TPR - 1	0 - 10
`acktimer`	The connection acknowledgment timer length in (100 * `timeinterval`) ms.	2	> 0
`rsptimer`	The response acknowledgment timer length in (100 * `timeinterval`) ms.	2	> 0
`polltimer`	The connection poll timer length in (100 * `timeinterval`) ms.	4	> 0
`rejecttimer`	The connection reject timer length in (100 * `timeinterval`) ms.	6	> 0

Parameter	Description	Default	Range
rembusytimer	The connection remote busy timer length in (100 * timeinterval) ms.	8	> 0
inacttimer	The connection inactivity timer length in (100 * timeinterval) ms.	30	> 0
maxretry	The maximum number of retries of an action on a connection.	6	0 - 100
xmitwindowsz	The maximum number of unacknowledged I-format protocol data units that can be transmitted on a connection before awaiting an acknowledgment.	14	0 - 127
rcvwindowsz	The maximum number of unacknowledged I-format protocol data units that can be received on a connection before an acknowledgment is sent.	14	0 - 127

Default values are set when the following conditions are true:

- The parameter is not set by the user.
- The user requests a default /etc/ildcf file built based on the adapters installed.
- The user codes a value of 0 for a parameter.

Timer Parameter Descriptions

acktimer

The acktimer parameter is used to manage the following sample sequences:

1. Attempting to establish, reset, or disconnect a connection.

```
SABME     start acknowledgment timer
 or   -------------------------------->
DISC
```

The acknowledgment timer expires before the receipt of a response.

```
SABME     start acknowledgment timer
 or   -------------------------------->
DISC
```

```
     stop acknowledgment timer
<-------------------------------- UA
```

2. Sending an FRMR in response to a received PDU of dubious distinction:

```
                     PDU with invalid N(R)
                             or
                     I PDU with invalid N(S)
                             or
<------------------- PDU of invalid length
                             or
                     unexpected UA PDU
                             or
                     response PDU with
                     invalid P/F setting

             start acknowledgment timer
FRMR  -------------------------------->
```

Acknowledgment timer expires before the receipt of a PDU.

```
            start acknowledgment timer
FRMR  -------------------------------->

  stop acknowledgment timer
                                   SABME, FRMR
<-------------------------------   DISC, or DM
```

3. There is also a special case of the acknowledgment timer, referred to in this implementation as the response acknowledgment timer (`rsptimer`). It is used when sending an I PDU.

```
   start response acknowledgement timer
I  ------------------------------------->
```

Response acknowledgment timer expires before the receipt of an acknowledgment.

```
          start poll timer
RR  -------------------------------->
```

polltimer — The `polltimer` parameter is used to manage situations where a Supervisory command PDU (RR, RNR, or REJ) is sent with the P/F bit set. This type of PDU is typically sent when:

- There has been a period of inactivity on a connection in information transfer mode.

- The remote node must be notified of a local busy condition occurring in information transfer mode.

The expiration of the poll timer causes another Supervisory command PDU (which may be of a different type than the first) to be sent with the P/F bit set, provided the retry count has not exceeded the maximum retry value. This timer, then, provides an extended retry mechanism for a connection in information transfer mode.

rejecttimer

The `rejecttimer` parameter controls the frequency with which a REJ PDU is sent to a remote node from which an I PDU with an unexpected N(S) was received and which has not corrected the situation by sending an I PDU with the expected N(S).

```
      <-----------------------  I PDU with
                                 unexpected N(S)
             start reject timer
    REJ  ------------------------>
```

Reject timer expires before the receipt of an I PDU with an expected N(S).

```
          start reject and poll timer
    REJ  ----------------------------->
          stop reject and poll timer
        <---------------------------   I PDU with
                                       expected N(S)
```

rembusytimer

The `rembusytimer` parameter is used to determine how long the local node should wait, after the remote node sends an RNR to indicate it is busy, before sending a Supervisory PDU with the P/F bit set to solicit the current state of the remote node. If the remote node indicates that it has cleared its busy condition before the timer expires, the local node stops the remote busy timer.

inacttimer

The `inacttimer` parameter controls how much time is allowed to elapse on a connection in information transfer mode between the issuing of command PDUs by the local node. If the inactivity timer expires because a command PDU has not been generated in the configured time interval, a Supervisory PDU with the P/F bit set is sent to the remote node to solicit its current state, provided that the connection is in information transfer mode. Each time a command PDU is sent by the local node, the inactivity timer is restarted.

The following rules of thumb should apply for the timer parameters:

- The `acktimer`, `rsptimer`, and `polltimer` parameters should have small relative values to allow for quick recovery from common transient error conditions on a connection.
- The `rejecttimer` and `rembusytimer` parameters should have intermediate relative values to allow the local and remote nodes time to recover without resorting to possibly unnecessary polling cycles.
- The `inacttimer` parameter should be set to a large relative value to provide a safety net in information transfer mode.

You may need to shift the values for the timer parameters to higher values if bridges are included in the network or a user application requires a substantial amount of time to respond to connection establishment requests or handle information flow.

Maximum Retry Parameter Description

The `maxretry` parameter determines the number of times a recovery operation is performed before notifying the user that an error has occurred on a connection. Typical examples of its use include the following:

- When the remote node fails to respond to a SABME sent by the local node to establish or reset the connection, the SABME is resent each time the acknowledgment timer expires, up to `maxretry` number of times.
- In information transfer mode, if the response acknowledgment timer expires after an I PDU has been sent, an RR with the P/F bit set is sent (and resent each time the poll timer expires) until the remote node responds or `maxretry` number of RRs have been sent.

In general, the `maxretry` value should not need to be large. Since the acknowledgment and poll timers are typically used in recovery operations that involve the `maxretry` parameter, the product of `maxretry` and either `acktimer`, `rsptimer`, or `polltimer` gives a rough estimate of the length of time allotted for the connection to attempt internal error recovery before notifying the user.

Window Size Parameter Descriptions

rcvwindowsz — The `rcvwindowsz` parameter is used to set the receive window size for I PDUs received locally on a connection. This value should agree with the transmit window size set for the connection at the remote node. If the local `rcvwindowsz` is greater than the remote transmit window size, I PDUs sent by the remote node are not acknowledged quickly. If the local `rcvwindowsz` is less than the remote transmit window size, there is a greater risk of the local node generating FRMR PDUs, requiring intervention by the user application when transient errors on the connection require

	the remote node to retransmit an I PDU. REJ PDUs are recovered internally.
xmitwindowsz	The `xmitwindowsz` parameter sets the local transmit window size for a connection. It denotes the number of unacknowledged I PDUs that the local node may have outstanding. The configured value should match the receive window size for the connection at the remote node, based on the same reasoning as for the `rcvwindowsz` parameter.

In many cases, the values assigned to `rcvwindowsz` and `xmitwindowsz` for adapters on a server node will depend on the transmit and receive window sizes specified for another LLC implementation on a client node. In cases where this LLC implementation is resident in both nodes, larger values for these parameters are useful in environments where much of the activity on a connection consists of file transfer operations. Smaller values are warranted if analysis of LLC2 connection component statistics reveals that connections are entering local or remote busy state frequently.

For a complete explanation of the keywords used, see the publication, *The Logical Link Control Driver for Solaris, Installation and Diagnostics*.

FILES

`/etc/llc2/default/llc2.*`

SEE ALSO

`llc2_autoconfig`(1), `llc2_config`(1), `llc2`(7D)

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 perform. 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 (separated by tabs):

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`(3C) 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`(3C)

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

mech, qop – mechanism and QOP files

SYNOPSIS

`/etc/gss/mech/etc/gss/qop`

DESCRIPTION

The `/etc/gss/mech` and `/etc/gss/qop` files contain tables showing installed security mechanisms and the Quality of Protection (QOP) associated with them, respectively. As security mechanisms are installed on the system, entries are added to these two files. Contents of these files may be accessed either manually (for example, with `cat`(1) or `more`(1)) or programmatically (with either `rpc_gss_get_mechanisms`(3NSL) or `rpc_gss_get_mech_info`(3NSL)).

The `/etc/gss/mech` file contains four fields:

mechanism name — ASCII string representing the mechanism.

object identifier — RPC OID for this mechanism.

shared library — Shared library which implements the services provided by this mechanism.

kernel module — Kernel module which implements the services provided by this mechanism.

The `/etc/gss/qop` file contains three fields:

QOP string — Name, in ASCII, of this Quality of Protection.

QOP value — Numeric value by which RPC identifies this QOP.

mechanism name — ASCII string representing the mechanism with which this QOP is associated.

EXAMPLES

EXAMPLE 1 A Typical Entry in `/etc/gss/mech`

This is a typical entry in a `/etc/gss/mech` file:

```
kerberosv5 1.2.840.113554.1.2.2 mech_krb5.so kmech_krb5
```

EXAMPLE 2 A Typical Entry in `/etc/gss/qop`

This is a typical entry in a `/etc/gss/qop` file:

```
GSS_KRB5_CONF_C_QOP_DES 0 kerberosv5
```

SEE ALSO

`rpc`(3NSL) , `rpc_gss_get_mechanisms`(3NSL) , `rpc_gss_get_mech_info`(3NSL) , `rpcsec_gss`(3NSL) , `attributes` *ONC+ Developer's Guide*

NAME

mnttab – mounted file system table

DESCRIPTION

The file `/etc/mnttab` is really a file system that provides read-only access to the table of mounted file systems for the current host. `/etc/mnttab` is read by programs using the routines described in `getmntent`(3C). Mounting a file system adds an entry to this table. Unmounting removes an entry from this table. Remounting a file system causes the information in the mounted file system table to be updated to reflect any changes caused by the remount. The list is maintained by the kernel in order of mount time. That is, the first mounted file system is first in the list and the most recently mounted file system is last. When mounted on a mount point the file system appears as a regular file containing the current `mnttab` information.

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. (See respective mount file system man page in SEE ALSO.)

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 file system in the form of *host:pathname*, or the name of a *swap file* (for example, a file made with `mkfile`(1M)).

IOCTLS

The following `ioctl`(2) calls are supported:

MNTIOC_NMOUNTS — Returns the count of mounted resources in the current snapshot in the `uint32_t` pointed to by *arg*.

MNTIOC_GETDEVLIST — Returns an array of `uint32_t`'s that is twice as long as the length returned by `MNTIOC_NMOUNTS`. Each pair of numbers is the major and minor device number for the file system at the corresponding line in the current `/etc/mnttab` snapshot. *arg* points to the memory buffer to receive the device number information.

MNTIOC_SETTAG — Sets a tag word into the options list for a mounted file system. A tag is a notation that will appear in the options string of a mounted file system but it is not recognized or interpreted by the file system code. *arg* points to a filled in `mnttagdesc` structure, as shown in the following example:

```
uint_t  mtd_major; /* major number for mounted fs */
uint_t  mtd_minor; /* minor number for mounted fs */
char    *mtd_mntpt; /* mount point of file system */
char    *mtd_tag;  /* tag to set/clear */
```

If the tag already exists then it is marked as set but not re-added. Tags can be at most `MAX_MNTOPT_TAG` long.

MNTIOC_CLRTAG — Marks a tag in the options list for a mounted file system as not set. *arg* points to the same structure as `MNTIOC_SETTAG`, which identifies the file system and tag to be cleared.

ERRORS

EFAULT — The arg pointer in an `MNTIOC_ ioctl` call pointed to an inaccessible memory location or a character pointer in a `mnttagdesc` structure pointed to an inaccessible memory location.

EINVAL — The tag specified in a `MNTIOC_SETTAG` call already exists as a file system option, or the tag specified in a `MNTIOC_CLRTAG` call does not exist.

ENAMETOOLONG — The tag specified in a `MNTIOC_SETTAG` call is too long or the tag would make the total length of the option string for the mounted file system too long.

WARNINGS

The `mnttab` file system provides the previously undocumented `dev=`*xxx* option in the option string for each mounted file system. This is provided for legacy applications that might have been using the `dev=information` option.

Using `dev=`*option* in applications is strongly discouraged. The device number string represents a 32-bit quantity and might not contain correct information in 64-bit environments.

Applications requiring device number information for mounted file systems should use the `getextmntent`(3C) interface, which functions properly in either 32- or 64-bit environments.

FILES

/etc/mnttab
Usual mount point for mnttab file system

/usr/include/sys/mntio.h
Header file that contains IOCTL definitions

SEE ALSO

mkfile(1M), mount_cachefs(1M), mount_hsfs(1M), mount_nfs(1M), mount_pcfs(1M), mount_ufs(1M), mount(1M), ioctl(2), read(2), poll(2), stat(2), getmntent(3C)

NOTES

The snapshot of the mnttab information is taken any time a read(2) is performed at offset 0 (the beginning) of the mnttab file. The file modification time returned by stat(2) for the mnttab file is the time of the last change to mounted file system information. A poll(2) system call requesting a POLLRDBAND event can be used to block and wait for the system's mounted file system information to be different from the most recent snapshot since the mnttab file was opened.

NAME

nca.if – the NCA configuration file that specifies physical interfaces

SYNOPSIS

`/etc/nca/nca.if`

DESCRIPTION

Specify the physical interfaces for which the Solaris Network Cache and Accelerator (“NCA”) feature will be configured in the `nca.if` configuration file. List the physical interfaces in the file, one per line. To configure NCA to listen on all physical interfaces present on the system backed by a `hostname.{interface_name}`, then list only an asterik (“*”) in `nca.if`.

When `ncakmod`(1) is invoked during system boot, it will attempt to `ifconfig`(1M) each physical interface specified in the `nca.if` file. Note that there must be an accompanying `hostname.{interface_name}` file and an entry in `/etc/hosts` for the contents of `hostname.{interface_name}`.

You must reboot in order to implement changes to the `nca.if` file.

EXAMPLES

IA

EXAMPLE 1 IA: `nca.if` on IA

The following is an example of an `nca.if` file that would be used on an IA system:

```
iprb1
iprb6
iprb8
```

SPARC

EXAMPLE 2 `nca.if` on SPARC

The following is an example of an `nca.if` file that would be used on a SPARC system:

```
hme2
hme3
hme4
```

All Platforms

EXAMPLE 3 Configuring NCA to Listen on All Physical Interfaces

The following example shows the contents of an `nca.if` file that would be used to configure either platform to listen on all physical interfaces present on the system:

```
*
```

FILES

`/etc/nca/nca.if`	Lists the physical interfaces on which NCA will run.
`/etc/hostname.{}{0-9}`	Lists all physical interfaces configured on the server.

`/etc/hosts` Lists all host names associated with the server. Entries in this file must match with entries in `/etc/hostname.{}{0-9}` for NCA to function.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWncar
Interface Stability	Evolving

SEE ALSO

`ifconfig`(1M), `nca`(1), `attributes`(5)

System Administration Guide, Volume 3

NAME

ncakmod.conf – the ncakmod configuration file

SYNOPSIS

/etc/nca/ncakmod.conf

DESCRIPTION

The ncakmod.conf file is used to configure the Solaris Network Cache and Accelerator (“NCA”) kernel module. The file contains two fields, key and value.

The status key is used to indicate if the user wants to have NCA turned on as a feature. If the value of status key is enabled, then the NCA kernel module will be pushed on to the specified interfaces. If the value of the status key is disabled, then the NCA kernel module will not be pushed on to any interfaces . The default is “disabled”.

The httpd_door_path key specifies the path name of the Solaris Door RPC mechanism that will be used to communicate with the http daemon. The default value is /var/run/nca_httpd_1.door.

In order to implement changes to the ncakmod.conf file, you will need to reboot.

EXAMPLES

EXAMPLE 1 A Sample ncakmod.conf File

The following is a sample ncakmod.conf file:

```
#
# NCA Kernel Module Configuration File
#
status=disabled
httpd_door_path=/var/run/nca_httpd_1.door
```

FILES

/etc/nca/ncakmod.conf The NCA kernel module configuration file.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWncar
Interface Stability	Evolving

SEE ALSO

nca(1), door_create(3DOOR), nca.if(4), attributes(5)

System Administration Guide, Volume 3

NAME

ncalogd.conf – the NCA logging configuration file

SYNOPSIS

/etc/nca/ncalogd.conf

DESCRIPTION

The ncalogd.conf is used to configure Solaris Network Cache and Accelerator ("NCA") logging. The file contains two fields, key and value.

The status key is used to indicate if the user wants to have NCA logging turned on. If the value of status key is enabled, then NCA logging will be turned on. If the value of the status key is disabled, then NCA logging will not be invoked. The default value is "disabled".

The logd_path_name key specifies the location of the log file. The value of logd_path_name is the absolute path to the log file. The default value is /var/nca/log. logd_path_name can also contain a white space delimited list of values for multiple log files to a maximum of 16. NCA logging moves to the next file on the list once the file size specified by logd_file_size has been reached. When the last file is full, NCA logging rotates back to the first file in the list. A pointer to the current log file is stored in /var/nca/current.

The logd_file_size key specifies the value of the file size, in bytes, allowed for each log file specified in by the logd_path_name key. The default value is 1000000 bytes.

In order to implement changes to the ncalogd.conf file, you will need to stop and start NCA logging or reboot.

EXAMPLES

EXAMPLE 1 A Sample ncalogd.conf File

The following is a sample ncalogd.conf file that specifies three log files:

```
#
# NCA Log Daemon Configuration File
#

status=disabled
logd_path_name=/var/nca/log1 /var/nca/log2 /var/nca/log3
logd_file_size=1000000
```

FILES

/etc/nca/ncalogd.conf	Lists configuration parameters for NCAlogging.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWncar
Interface Stability	Evolving

SEE ALSO

nca(1), door_create(3X), attributes(5)

System Administration Guide, Volume 3

NAME

ndpd.conf – configuration file for IPv6 router autoconfiguration

SYNOPSIS

`/etc/inet/ndpd.conf`

DESCRIPTION

The `ndpd.conf` file contains configuration information for `in.ndpd`()1M when used on a router. This file does not need to exist or can be empty on a host. The file has one configuration entry per line; note that lines can be extended with "\" followed by a newline. There are four forms of configuration entries which are identified by the first field on the line: `ifdefault`, `prefixdefault`, `if`, or `prefix`. The `ifdefault` and `if` entries set interface configuration variables; the former establishes the defaults for all interfaces. Any `ifdefault` entries must precede any `if` entries in the file.

The `prefixdefault` and `prefix` entries control per-prefix configuration variables. `prefixdefault` establishes the defaults for all prefixes on all interfaces. Any `prefixdefault` entries must precede any `prefix` entries in the file.

Each `ifdefault` entry is composed of a single line of the form:

```
ifdefault [ if-variable-name value ]*
```

Each `if` entry is composed of a single line of the form:

```
if interface [ if-variable-name value ]*
```

Each `prefixdefault` entry is composed of a single line of the form:

```
prefixdefault [ prefix-variable-name value ]*
```

Each prefix entry is composed of a single line of the form:

```
prefix prefix/prefix_length interface [ prefix-variable-name value ]*
```

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.

`interface` — The name of a network interface, for example, `le0`.

`prefix` — An IPv6 address in standard hexadecimal notation, for example, `fec0:0:0:1::0`.

`prefix_length` — A number between 0 and 128.

`if-variable-name`

An interface variable as discussed in *RFC 2461* and *RFC 2462*. The following lists the each interface variable and its default value and unit:

Variable Name	Default	Unit
DupAddrDetectTransmits	1	Counter
AdvSendAdvertisements	false	Boolean
MaxRtrAdvInterval	600	Seconds
MinRtrAdvInterval	200	Seconds
AdvManagedFlag	false	Boolean
AdvOtherConfigFlag	false	Boolean
AdvLinkMTU	0	Bytes
AdvReachableTime	0	Milliseconds
AdvRetransTimer	0	Milliseconds
AdvCurHopLimit	0	Counter
AdvDefaultLifetime	1800	Seconds

`prefix-variable-name`

A prefix variable as discussed in *RFC 2461* and *RFC 2462*. The following lists the each interface variable and its default value and unit:

Variable Name	Default	Unit
AdvValidLifetime	2592000	Seconds
AdvOnLinkFlag	true	Boolean
AdvPreferredLifetime	604800	Seconds
AdvAutonomousFlag	true	Boolean
AdvValidExpiration	not set	Date/Time
AdvPreferredExpiration	not set	Date/TIme

The “Expiration” variables are used to specify that the lifetime should be decremented in real time as specified in *RFC 2461*. If an "Expiration" variable is set then it takes precedence over the corresponding "Lifetime" variable setting.

`value` The value is a function of the unit. Boolean values are `true`, `false`, `on`, off, `1`, or `0`.

Values in seconds can have characters appended for day (`d`), hour `h`), minute (`m`) and second (`s`). The default is seconds. For example, `1h` means 1 hour. This is equivalent to the value `3600`.

Values in milliseconds can have characters appended for day (`d`), hour (`h`), minute (`m`) second (`s`), and millisecond (`ms`). The default is milliseconds. For example, `1h` is equivalent to the value `3600000`.

Date/time values are strings that use the recommended ISO date format described as "`%Y-%m-%d %R`", which represents a 4 digit year, a dash character, a numeric month, a dash character, and a numeric day of the month, followed by one or more whitespace characters and finally a 24 hour clock with hours, a colon, and minutes. For example, `1999-01-31 20:00` means 8pm January 31 in 1999. Since the date/time values contain a space, use single or double quotes to declare the value. For example:

```
prefixdefault AdvPreferredExpiration '1999-01-31 20:00'
```

EXAMPLES

EXAMPLE 1 Sending Router Advertisements for all Interfaces

The following example can be used to send router advertisements out to all interfaces:

```
 # Send router advertisements out all interfaces
  ifdefault AdvSendAdvertisements on AdvOnLinkFlag on AdvAutonomousFlag on

  # Advertise a (bogus) global prefix and a site
  # local prefix on three interfaces using the default lifetimes
  prefix 2:0:0:9255::0/64  hme0
  prefix fec0:0:0:9255::0/64 hme0

  prefix 2:0:0:9256::0/64  hme1
  prefix fec0:0:0:9256::0/64 hme1

  prefix 2:0:0:9259::0/64  hme2
  prefix fec0:0:0:9259::0/64 hme2
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsr

SEE ALSO

`in.ndpd`(1M), `attributes`(5), `icmp6`(7P), `ip6`(7P)

Narten, T., Nordmark, E., and Simpson, W., *RFC 2461, Neighbor Discovery for IP Version 6 (IPv6)*, The Internet Society, December 1998.

Thomson, S., and Narten, T., *RFC 2462, IPv6 Stateless Address Autoconfiguration*, The Internet Society, December 1998.

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`(3NSL)) are part of the Network Selection component. The Network Selection component also includes `getnetpath`(3NSL) 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 IDs*; 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, and the like.
`inet6`	Internetwork over IPv6: UDP, TCP, and the like.
`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, and the like.
`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 IDs*; 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(3DL).

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

EXAMPLE 1 A Sample netconfig File

Below is a sample netconfig file:

```
#
#  The "Network Configuration" File.
#
#  Each entry is of the form:
#
#   <networkid> <semantics> <flags> <protofamily> <protoname><device> \
#         <nametoaddrlibs>
#
#  The "-" in <nametoaddrlibs> 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 netdirgetbyname, netdirgetbyaddr,
#  gethostbyname, gethostbyaddr, getservbyname, and getservbyport.
#  There are no nametoaddrlibs for the inet family in Solaris anymore.
#
#
#  The following two entries starting with udp6 and tcp6 are meant to be
#  used for IPv6. If you have Ipv6 enabled on your machine then you can
#  uncomment these two lines to enable RPC and NFS to use the Ipv6 stack.
#  Consult your network administrator before uncommenting.
#
#udp6       tpi_clts      v     inet6   udp    /dev/udp6       -
#tcp6       tpi_cots_ord  v     inet6   tcp    /dev/tcp6       -

udp       tpiclts      v    inet      udp  /dev/udp       -
tcp       tpicotsord   v    inet      tcp  /dev/tcp       -
rawip     tpiraw       -    inet      -    /dev/rawip     -
ticlts    tpiclts      v    loopback  -    /dev/ticlts    straddr.so
ticotsord tpicotsord   v    loopback  -    /dev/ticotsord straddr.so
ticots    tpicots      v    loopback  -    /dev/ticots    straddr.so
```

FILES

<netconfig.h>

SEE ALSO

`dlopen`(3DL), `getnetconfig`(3NSL), `getnetpath`(3NSL), `nsswitch.conf`(4)

NFS Administration Guide

Transport Interfaces Programming Guide

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`(3C), `hosts`(4), `hosts.equiv`(4), `nsswitch.conf`(4), `passwd`(4), `shadow`(4)

NOTES

`netgroup` requires NIS or NIS+.

Applications may make general membership tests using the `innetgr()` function (see `innetgr`(3C)).

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.

Use of placeholders will improve search performance.

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`(3NSL)) 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

EXAMPLE 1 A sample `netid` file.

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`(3NSL), `secure_rpc`(3NSL), `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 subnetting. It supports both standard subnetting as specified in *RFC-950* and variable length subnetting as specified in *RFC-1519*. When using standard subnetting there should be a single line for each network that is subnetted 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.

When using variable length subnetting, the format is identical. However, there should be a line for each subnet with the first field being the subnet and the second field being the netmask that applies to that subnet. The users of the database, such as `ifconfig`(1M), perform a lookup to find the longest possible matching mask. It is possible to combine the *RFC-950* and *RFC-1519* form of subnet masks in the netmasks file. For example,

```
128.32.0.0 255.255.255.0
128.32.27.0 255.255.255.240
128.32.27.16 255.255.255.240
128.32.27.32 255.255.255.240
128.32.27.48 255.255.255.240
128.32.27.64 255.255.255.240
128.32.27.80 255.255.255.240
128.32.27.96 255.255.255.240
128.32.27.112 255.255.255.240
128.32.27.128 255.255.255.240
128.32.27.144 255.255.255.240
128.32.27.160 255.255.255.240
128.32.27.176 255.255.255.240
128.32.27.192 255.255.255.240
128.32.27.208 255.255.255.240
128.32.27.224 255.255.255.240
128.32.27.240 255.255.255.240
128.32.64.0 255.255.255.192
```

can be used to specify different netmasks in different parts of the 128.32.0.0 Class B network number. Addresses 128.32.27.0 through 128.32.27.255 have a subnet mask with 28 bits in the combined network and subnet fields (often referred to as the subnet field) and 4 bits in the host field. Furthermore, addresses 128.32.64.0 through 128.32.64.63 have a 26 bits in the subnet field. Finally, all other addresses in the range 128.32.0.0 through 128.32.255.255 have a 24 bit subnet field.

Invalid entries are ignored.

SEE ALSO

`ifconfig`(1M), `inet`(7P)

Postel, Jon, and Mogul, Jeff, *Internet Standard Subnetting Procedure*, RFC 950, Network Information Center, SRI International, Menlo Park, Calif., August 1985.

V. Fuller, T. Li, J. Yu, K. Varadhan, *Classless Inter-Domain Routing (CIDR): an Address Assignment and Aggregation Strategy*, RFC 1519, Network Information Center, SRI International, Menlo Park, Calif., September 1993.

T. Pummill, B. Manning, *Variable Length Subnet Table For IPv4*, RFC 1878, Network Information Center, SRI International, Menlo Park, Calif., December 1995.

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

EXAMPLE 1 A Sample `.netrc` File

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

SYNOPSIS

/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(3SOCKET) 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(7P). Network names may contain any printable character other than a field delimiter, NEWLINE, or comment character.

SEE ALSO

getnetbyaddr(3SOCKET), getnetbyname(3SOCKET), inet(3SOCKET), nsswitch.conf(4), inet(7P)

NOTES

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

The network database does not support subnet masks in general, so getnetbyaddr(3SOCKET) cannot differentiate between networks of 11.128.0.0 / 255.192.0.0 and 11.128.0.0 / 255.240.0.0.

NAME

nfslog.conf – NFS server logging configuration file

SYNOPSIS

/etc/nfs/nfslog.conf

DESCRIPTION

The nfslog.conf file specifies the location of the NFS server logs, as well as the location of the private work files used by the NFS server and nfslogd(1M) daemon during logging. Each entry in the file consists of a mandatory tag identifier and one or more parameter identifiers. The parameter identifier specifies the value or location of the specific parameter. For instance, the parameter identifier "log=/var/nfs/logs/serverLog" specifies the location of the NFS server activity log. The mandatory tag identifier serves as an index into the /etc/nfs/nfslog.conf file to identify the various parameters to be used. At export time, the share_nfs(1M) command specifies the NFS server logging parameters to use by associating a tag from the /etc/nfs/nfslog.conf file to the exported file system. It is legal for more than one file system to be exported using the same logging tag identifier.

A "global" tag identifier is included in /etc/nfs/nfslog.conf. It specifies the default set of values to be used during logging. If no tag identifier is specified at export time, then the values in the "global" entry are used. The "global" values can be modified by updating this entry in /etc/nfs/nfslog.conf.

Each entry in the file must contain a mandatory tag identifier and at least one parameter / value pair. If a parameter is not specified in a given entry, the global value of the parameter will be used. The exact entry syntax follows:

```
<tag> [defaultdir=<path>] [log=<path><file>] [fhtable=<path><file>] \
 [buffer=<path><file>] [logformat=basic|extended]
```

defaultdir=<path>
: Specifies the directory where the logging files and working files will be placed. This path is prepended to all relative paths specified in other parameters.

log=<path><file>
: Specifies the location of the user-readable log file. The log will be located in the defaultdir, unless <path> is an absolute path.

fhtable=<path><file>
: Specifies the location of the private file handle to path mapping database files. These database files are for the private use of the NFS server kernel module and the nfslogd daemon. These files will be located in the defaultdir, unless <path> is an

absolute path. These database files are permanently stored in the file system. Consult nfslogd(1M) for information on pruning the database files.

buffer=<path><file>
: Specifies the location of the private work buffer file used by the NFS server kernel module to record raw RPC information. This file is later processed by the nfslog daemon, which in turn generates the user-readable log file. This work buffer file will be located in the defaultdir, unless <path> is an absolute path.

logformat=basic|extended
: Sets the format of the user-readable log file. If not specified, the basic format is used. The basic format is compatible with log files generated by the Washington University FTPd. The extended format provides a more detailed log, which includes directory modification operations not included in the basic format, such as mkdir, rmdir and remove. Note that the extended format is not compatible with Washington University's FTPd log format.

EXAMPLES

EXAMPLE 1 Using the global Tag

The "global" tag may be modified so that all exported file systems that enabled logging use a common set of parameters that conform to the specific needs of the user. These values are used until a specific tag identifier overrides them.

```
global defaultdir=/var/nfs log=logs/nfslog \
    fhtable=tables/fhtable buffer=buffers/nfslog_workbuffer \
    logformat=basic
```

EXAMPLE 2 Overriding the Global defaultdir and logformat

Because log files can become very large, it may be desirable to store the logs and working files in separate file systems. This can be easily accomplished by simply specifying a different defaultdir for every file system exported by means of a unique tag:

```
engineering defaultdir=/engineering/logging \
             logformat=extended
accounting defaultdir=/accounting/logging
marketing  defaultdir=/marketing/logging
```

File systems shared with the engineering identifier will have their logs and workfiles located in /engineering/logging. For instance, the log file will be located at /engineering/logging/logs/nfslog. Note that the engineering log file will be stored in the extended format, while the rest of the log files will remain in the basic format.

Any of the parameters can be updated in a tag identifier, which overrides the global settings.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsr

SEE ALSO

nfslogd(1M), share_nfs(1M), attributes(5)

NOTES

Logs, work files, and file handle to path mapping database can become very large. Be aware of appropriate placement within the file system name space. See nfslogd(1M)) for information on pruning the database files and cycling logs.

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 table 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 subdomain `groups_dir.`*domain-name* as *group* objects. These two tables appear as files in the `/var/nis/data` 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 (see `nistbladm`(1)). 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 that 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`
: 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`
: Contains the current cache of NIS+ bindings being maintained by the cache manager. The contents can be viewed with `nisshowcache`(1M).

`client_info`
: Contains configuration information (preferred servers, options, etc.) for `nis_cachemgr`(1M) and (potentially) other NIS+ clients on the system. It is manipulated by the `nisprefadm`(1M) command.

`.pref_servers`	A cached copy of preferred server information. It is maintained by `nis_cachemgr.` Do not edit this file manually.
`trans.log`	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.
`data.dict`	A dictionary that is used by the NIS+ database to locate its files. It is created by the default NIS+ database package.
`data.dict.log`	The log file for the database dictionary. When the server is checkpointed (see the `–C` option of `nisping`(1M)), this file will be deleted.
`data`	Contains databases that the server uses.
`data/root.object`	On root servers, this file contains a directory object that describes the root of the name space.
`data/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.
`data/`*table_name*	For each table in the directory there is 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*.
`data/`*table_name.log*	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 data / *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` operations periodically (such as, once a day) to checkpoint the log file. This can be done either through a `cron`(1M) job, or manually.

`data/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`.
`data/cred.org_dir`	Table containing the credentials of principals in this NIS+ domain.
`data/groups_dir`	Table containing the group authorization objects needed by NIS+ to authorize group access.
`data/serving_list`	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), `nis`(1), `nis_cachemgr`(1M), `niscat`(1), `nismatch`(1), `nistbladm`(1), `rm`(1), `cron`(1M), `nisclient`(1M), `nisinit`(1M), `nislog`(1M), `nisping`(1M), `nisprefadm`(1M), `nisshowcache`(1M), `nis_objects`(3NSL)

NAME

nologin – message displayed to users attempting to log on in the process of a system shutdown

SYNOPSIS

`/etc/nologin`

DESCRIPTION

The `/etc/nologin` file contains the message displayed to users attempting to log on to a machine in the process of being shutdown. After displaying the contents of the `nologin` file, the `login` procedure terminates, preventing the user from logging onto the machine.

This procedure is preferable to terminating a user's session by `shutdown` shortly after the user has logged on.

Logins by super-user are not affected by this procedure.

The message contained in the `nologin` file is editable by super-user. A typical `nologin` file contains a message similar to:

```
NO LOGINS: System going down in 10 minutes.
```

SEE ALSO

`login`(1), `rlogin`(1), `telnet`(1), `shutdown`(1M)

NAME

note – specify legal annotations

SYNOPSIS

/usr/lib/note

DESCRIPTION

Each file in this directory contains the NOTE (also _NOTE) annotations legal for a single tool. The name of the file, by convention, should be the tool vendor's stock name, followed by a hyphen, followed by the tool name. For example, for Sun's lock_lint tool the filename should be SUNW-lock_lint.

The file should contain the names of the annotations understood by the tool, one per line. For example, if a tool understands the following annotations:

```
NOTE(NOT_REACHED)
NOTE(MUTEX_PROTECTS_DATA(list_lock, list_head))
```

then its file in /usr/lib/note should contain the entries:

```
NOT_REACHED
MUTEX_PROTECTS_DATA
```

Blank lines, and lines beginning with a pound (#), are ignored.

While /usr/lib/note is the default directory tools search for such files, they can be made to search other directories instead simply by setting environment variable NOTEPATH to contain the paths, separated by colons, of directories to be searched, e.g., /usr/mytool/note:/usr/lib/note.

USAGE

These files are used by such tools whenever they encounter NOTEs they do not understand. If a file in /usr/lib/note contains the annotation, then it is valid. If no such file contains the annotation, then the tool should issue a warning complaining that it might be invalid.

ENVIRONMENT VARIABLES

NOTEPATH — specify paths to be searched for annotation files. Paths are separated by colons (":").

SEE ALSO

NOTE(3EXT)

NAME

nscd.conf – name service cache daemon configuration

SYNOPSIS

`/etc/nscd.conf`

DESCRIPTION

The `nscd.conf` file contains the configuration information for `nscd`(1M). Each line specifies either an *attribute* and a *value*, or an *attribute*, *cachename*, and a *value*. Fields are separated either by 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 `nscd`.

cachename is represented by `hosts`, `ipnodes`, `passwd`, or `groups`.

attribute supports the following:

`logfile` *debug-file-name*
: Specifies name of the file to which debug info should be written. Use `/dev/tty` for standard output.

`debug-level` *value*
: Sets the debug level desired. *value* may range from `0` (the default) to `10`. Use of this option causes `nscd`(1M) to run in the foreground and not become a daemon. Note that the output of the debugging command is not likely to remain the same from release-to-release; scripts should *not* rely on its format.

`enable-cache` *cachename value*
: Enables or disables the specified cache. *value* may be either `yes` or `no`.

`positive-time-to-live` *cachename value*
: Sets the time-to-live for positive entries (successful queries) in the specified cache. *value* is in integer seconds. Larger values increase cache hit rates and reduce mean response times, but increase problems with cache coherence. Note that sites that push (update) NIS maps nightly can set the value to be the equivalent of 12

hours or more with very good performance implications.

`negative-time-to-live` *cachename value*
: Sets the time-to-live for negative entries (unsuccessful queries) in the specified cache. *value* is in integer seconds. Can result in significant performance improvements if there are several files owned by uids (user IDs) not in system databases; should be kept small to reduce cache coherency problems.

`suggested-size` *cachename value*
: Sets the suggested number of hash buckets in the specified cache. This parameter should be changed only if the number of entries in the cache exceeds the suggested size by more than a factor of four or five. Since this is the internal hash table size, *value* should remain a prime number for optimum efficiency.

`keep-hot-count` *cachename value*
: This attribute allows the administrator to set the number of entries `nscd`(1M) is to keep current in the specified cache. *value* is an integer number which should approximate the number of entries frequently used during the day.

`check-files` *cachename value*
: Enables or disables checking the file belonging to the specified *cachename* for changes. If enabled (which is the default), changes in the corresponding file cause the cache to be invalidated within 10 seconds. Can be disabled if files are never modified for a slight performance boost, particularly

over NFS. *value* may be either `yes` or `no`.

SEE ALSO

`nscd`(1M), `group`(4), `hosts`(4), `ipnodes`(4), `passwd`(4)

WARNINGS

The `nscd.conf` interface is included in this release on an uncommitted basis only and is subject to change or removal in a future minor release.

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, ipnodes, `passwd`/`shadow`), and groups. Data for these can come from a variety of sources: host-names and host-addresses, for example, may be found in `/etc/hosts`, NIS, NIS+, LDAP, or DNS. Zero 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` file:

Database	*Used By*
`aliases`	`sendmail`(1M)
`automount`	`automount`(1M)
`bootparams`	`rpc.bootparamd`(1M)
`ethers`	`ethers`(3SOCKET)
`group`	`getgrnam`(3C)
`hosts`	`gethostbyname`(3NSL). See `Interaction with netconfig`.
`ipnodes`	`getipnodebyname`(3SOCKET)
`netgroup`	`innetgr`(3C)
`netmasks`	`ifconfig`(1M)
`networks`	`getnetbyname`(3SOCKET)
`passwd`	`getpwnam`(3C), `getspnam`(3C)
`printers`	`lp`(1), `lpstat`(1), `cancel`(1), `lpr`(1B), `lpq`(1B), `lprm`(1B), `in.lpd`(1M), `lpadmin`(1M), `lpget`(1M), `lpset`(1M)
`protocols`	`getprotobyname`(3SOCKET)
`publickey`	`getpublickey`(3NSL), `secure_rpc`(3NSL)
`rpc`	`getrpcbyname`(3NSL)
`sendmailvars`	`sendmail`(1M)
`services`	`getservbyname`(3SOCKET).
	See `Interaction with netconfig`.

The following sources may be used:

Source	*Uses*
`files`	`/etc/hosts`, `/etc/passwd`, `/etc/inet/inodes`, `/etc/shadow`
`nis`	NIS(YP)
`nisplus`	NIS+
`ldap`	LDAP
`dns`	Valid only for `hosts`; uses the Internet Domain Name Service.
`compat`	Valid only for `passwd` and `group`; implements "+" and "-". See `Interaction with +/- syntax`.
`user`	Valid only for printers; implements support for `${HOME}/.printers`.
`xfn`	Valid only for printers; implements support for FNS printer contexts. Provided to allow transition away from FNS printer contexts.

There is an entry in `/etc/nsswitch.conf` for each database. Typically these entries will be simple, such as "`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 configured on this system or internal failure.
`NOTFOUND`	Source responded “`no such entry`”
`TRYAGAIN`	Source is busy or not responding, 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.

Additionally, for `TRYAGAIN` only, the following actions are possible:

Action	*Meaning*
`forever`	Retry the current source forever.
n	Retry the current source *n* more times, where *n* is an integer between `0` and `MAX_INT` (that is, 2.14 billion). After *n* retries has been exhausted, the action will continue to the next source.

The complete syntax of an entry is:

```
<entry>     ::= <database> ":" [<source>
[<criteria>]]*
<criteria>  ::= "[" <criterion>+ "]"
<criterion> ::= <status> "=" <action>
<status>    ::= "success" | "notfound" | "unavail" | "tryagain"
```

For every status except `TRYAGAIN`, the action syntax is:

```
<action>    ::= "return"  | "continue"
```

For the `TRYAGAIN` status, the action syntax is:

```
<action>    ::= "return"  | "continue" | "forever" | <n>
<n>         ::= 0...MAX_INT
```

Each entry occupies a single line in the file. Lines that are blank, or that start with white space, 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 for DNS and the NIS server in “DNS-forwarding mode” (and DNS server not responding or busy) is [`SUCCESS`=return `NOTFOUND`=continue `UNAVAIL`=continue `TRYAGAIN`=continue].

The default criteria for all other sources is [`SUCCESS`=return `NOTFOUND`=continue `UNAVAIL`=continue `TRYAGAIN`=forever].

The default, or explicitly specified, criteria are meaningless following the last source in an entry; and they 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`(3NSL), `getipnodebyname`(3SOCKET), `getservbyname`(3SOCKET), and `netdir_getbyname`(3NSL) functions are all implemented in terms of the same internal library function. This function obtains the system-wide source lookup policy for `hosts`, `ipnodes`, 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 nametoaddr libraries. See the NOTES section in `gethostbyname`(3NSL) and `getservbyname`(3SOCKET) for details.

Interaction with NIS+ 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 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.

In SunOS 5.3 (Solaris 2.3) and compatible versions, the NIS+ server in "NIS/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 Password Aging

When password aging is turned on, only a limited set of possible name services are permitted for the `passwd`: database in the `/etc/nsswitch.conf` file:

passwd:	files
passwd:	files nis
passwd:	files nisplus
passwd:	files ldap
passwd:	compat

passwd_compat:	nisplus
passwd_compat:	ldap

Any other settings will cause the `passwd`(1) command to fail when it attempts to change the password after expiration and will prevent the user from logging in. These are the *only* permitted settings when password aging has been turned on. Otherwise, you can work around incorrect `passwd`: lines by using the `-r repository` argument to the `passwd`(1) command and using `passwd -r repository` to override the `nsswitch.conf` settings and specify in which name service you want to modify your password.

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 NIS/YP compatibility 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" or “ldap” 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 NIS/YP compatibility 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
ipnodes:	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
printers:	user files nis nisplus xfn

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 LDAP is the enterprise level name service, the default configuration should be modified to use `ldap` instead of `nis` for every database on client machines. The file `/etc/nsswitch.ldap` 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 OR ldap
group:	compat
group_compat:	nisplus OR ldap

In order to get information from the Internet Domain Name Service for hosts that are not listed in the enterprise level name service, NIS+ or LDAP, 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

or

hosts:	ldap 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 5,000 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.
/usr/lib/nss_ldap.so.1	Implements "ldap" source.
/usr/lib/nss_user.so.1	Implements "user" source.
/usr/lib/nss_xfn.so.1	Implements "xfn" source.
/etc/netconfig	Configuration file for netdir(3NSL) functions that redirects hosts / devices 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".
/etc/nsswitch.ldap	Sample configuration file that uses "files" and "ldap".
/etc/nsswitch.dns	Sample configuration file that uses “files” and “dns” (but only for hosts:).

SEE ALSO

ldap(1), nis+(1), passwd(1), automount(1M), ifconfig(1M), rpc.bootparamd(1M), rpc.nisd(1M), sendmail(1M), ethers(3SOCKET), getgrnam(3C), gethostbyname(3NSL), getipnodebyname(3SOCKET), getnetbyname(3SOCKET), getnetgrent(3C), getprotobyname(3SOCKET), getpublickey(3NSL), getpwnam(3C), getrpcbyname(3NSL), getservbyname(3SOCKET), getspnam(3C), netdir(3NSL), secure_rpc(3NSL), 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. The same applies for using ldap along with nis or nisplus.

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), `shadow`(4).

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 – 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/platform 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 separated 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.
SPOOLEDSIZE*	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

EXAMPLE 1 A sample .packagetoc file.

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

packingrules – packing rules file for cachefs and filesync

SYNOPSIS

`$HOME/.packingrules`

DESCRIPTION

`$HOME/.packingrules` is a packing rules file for `filesync` and `cachefspack`. `$HOME/.packingrules` contains a list of directories and files that are to be packed and synchronized. It also contains a list of directories and files that are to be specifically excluded from packing and synchronization. See `filesync`(1) and `cachefspack`(1M).

The `$HOME/.packingrules` file is automatically created if users invoke `filesync` with filename arguments. By using `filesync` options, users can augment the packing rules in `$HOME/.packingrules` .

Many users choose to manually create the packing rules file and edit it by hand. Users can edit `$HOME/.packingrules` (using any editor) to permanently change the `$HOME/.packingrules` file, or to gain access to more powerful options that are not available from the command line (such as `IGNORE` commands). It is much easier to enter complex wildcard expressions by editing the `$HOME/.packingrules` file.

Blank lines and lines that begin with a pound sign (‘#’) are ignored.

Any line can be continued by placing a backslash (‘\’) immediately before the NEWLINE.

All other lines in the `$HOME/.packingrules` file have one of the following formats:

`PACKINGRULES`	*major*. *minor*. This line is not actually required, but it should be the first line of every packing rules file. This line identifies the packing rules file for the `file`(1) command and specifies a format version number. The current version number is 1.1. See `file`(1).
BASE *directory-1* [*directory-2*]	This line identifies a directory (or pair of directories) under which files should be packed and synchronized. At least one directory name must be specified. For rules that are to be used by `filesync` a second directory name (where the copies are to be kept) must also be specified. The arguments must be

fully qualified path names, and may include environment variables.

`LIST` *name* ...
: This line enumerates a list of files and sub-directories (beneath the current `BASE`) that are to be kept synchronized. This specification is recursive, in that specifying the name of a directory automatically includes all files and subdirectories it contains. Regular expressions (as described in `glob` and `gmatch`) are permitted. See `glob`(1) and `gmatch`(3GEN).

`IGNORE` *name* ...
: This line enumerates a list of files that are not to be kept synchronized. Regular expressions (using `glob` and `gmatch`) are permitted.

There are important differences between the arguments to `LIST` and `IGNORE` statements. The arguments to a `LIST` statement can contain slashes and are interpreted as file names relative to the `BASE` directories. The arguments to an `IGNORE` statement are simpler names or expressions that cannot contain slashes. An `IGNORE` statement will not override a `LIST` statement. `IGNORE` statements only exclude files that are found beneath `LISTed` directories.

If the first name argument to a `LIST` statement begins with an exclamation point (‘!’), the remainder of the statement will be executed as a command. The command will be run in the current `BASE` directory. The output of the command will be treated as a list of newline separated file names to be packed/synchronized. The resulting file names will be interpreted relative to the enclosing `BASE` directory.

If the first name argument to an `IGNORE` statement begins with an exclamation point (‘!’), the remainder of the statement will be executed as a command. The command will be run in the current `BASE` directory. The command will be expected to figure out which names should not be synchronized. The output of the command will be treated as a list of newline separated file names that should be excluded from the packing and synchronization list.

Commands will be broken into distinct arguments and run directly with `sh –c`. Blanks can be embedded in an argument by escaping them with a backslash (‘\’) or enclosing the argument in double quotes (‘ " ’). Double quotes can be passed in arguments by escaping the double quotes with a backslash (‘\’).

`LIST` lines only apply to the `BASE` statement that precedes them. `IGNORE` lines can appear before any `BASE` statement (in which case they apply to all `BASEs`) or after a BASE statement (in which case they only apply to the BASE that precedes them). Any number of these statements can occur in any combination. The order is not important.

EXAMPLES

EXAMPLE 1 A sample `$HOME.packingrules` file.

The use of these statements is illustrated in the following `$HOME.packingrules` file.

```
#
# junk files, not worth copying
#
IGNORE core *.o *.bak *%
#
# most of the stuff I want to keep in sync is in my $HOME
#
BASE /net/bigserver/export/home/myname $HOME
# everything in my work sub-directory should be maintained
LIST work
# a few of my favorite mail boxes should be replicated
LIST m/incoming
LIST m/action
LIST m/pending
#
# I like to carry around a couple of project directories
# but skip all the postscript output
#
BASE /net/bigserver/export/projects $HOME/projects
LIST poindexter epiphany
IGNORE *.ps
#
# the foonly package should always be kept on every machine
#
BASE /net/bigserver/opt/foonly /opt/foonly
LIST !cat .packinglist
#
# and the latest executables for the standard build environment
#
BASE /net/bigserver/export/buildenv $HOME/buildenv
LIST !find . -type f -a -perm -111 -a -print
```

SEE ALSO

`file`(1), `filesync`(1), `cachefspack`(1M)

NAME

pam.conf – configuration file for pluggable authentication modules

SYNOPSIS

`/etc/pam.conf`

DESCRIPTION

`pam.conf` is the configuration file for the Pluggable Authentication Module architecture, or PAM. A PAM module provides functionality for one or more of four possible services: authentication, account management, session management, and password management. An authentication service module provides functionality to authenticate a user and set up user credentials. An account management module provides functionality to determine if the current user's account is valid. This includes checking for password and account expiration, as well as verifying access hour restrictions. A session management module provides functionality to set up and terminate login sessions. A password management module provides functionality to change a user's authentication token or password. Each of the four service modules can be implemented as a shared library object which can be referenced in the `pam.conf` configuration file.

Simplified PAM.CONF configuration file

The `pam.conf` file contains a listing of services. Each service is paired with a corresponding service module. When a service is requested, its associated module is invoked. Each entry has the following format:

service_name module_type control_flag module_path options

Below is an example of the `pam.conf` configuration file with support for authentication, account management, and session management modules.

```
login    auth    required  /usr/lib/security/$ISA/pam_unix.so.1  debug
login    session required  /usr/lib/security/$ISA/pam_unix.so.1
login    account required  /usr/lib/security/$ISA/pam_unix.so.1
telnet   session required  /usr/lib/security/$ISA/pam_unix.so.1
other    auth    required  /usr/lib/security/$ISA/pam_unix.so.1
other    passwd  required  /usr/lib/security/$ISA/pam_unix.so.1
```

The *service_name* denotes the service (for example, `login`, `dtlogin`, or `rlogin`). The keyword, *other*, indicates the module all other applications which have not been specified should use. The *other* keyword can also be used if all services of the same *module_type* have the same requirements. In the example above, since all of the services use the same session module, they could have been replace by a single *other* line.

module_type denotes the service module type: authentication (*auth*), account management (*account*), session management (*session*), or password management (*password*).

The *control_flag* field determines the behavior of stacking, and will be discussed in more detail below.

The module_path field specifies the pathname to a shared library object which implements the service functionality. If the pathname is not absolute, it is assumed to be relative to `/usr/lib/security/$ISA/`. If the pathname contains the `$ISA` token, that token is replaced by an implementation defined directory name which defines the path relative to the calling program's instruction set architecture.

The *options* field is used by the PAM framework layer to pass module specific options to the modules. It is up to the module to parse and interpret the options. This field can be used by the modules to turn on debugging or to pass any module specific parameters such as a TIMEOUT value. It can also be used to support unified login. The options supported by the modules are documented in their respective manual pages. For example, `pam_unix`(5) lists the options accepted by the UNIX module.

Integrating Multiple Authentication Services With Stacking

When a service_name of the same *module_type* is defined more than once, the service is said to be *stacked*. Each module referenced in the *module_path* for that service is then processed in the order that it occurs in the configuration file. The *control_flag* field specifies the continuation and failure semantics of the modules, and may be *requisite*, *required*, *optional*, or *sufficient*.

The PAM framework processes each service module in the stack. If all *requisite* and *required* modules in the stack succeed, then success is returned, and *optional* and *sufficient* error values are ignored. If one or more *requisite* or *required* modules fail, then the error value from the first *requisite* or *required* module that failed is returned.

If none of the service modules in the stack are designated as *requisite* or *required*, then the PAM framework requires that at least one *optional* or *sufficient* module succeed. If all fail then the error value from the first service module in the stack is returned.

The *requisite* and *sufficient* flags cause two exceptions to the above semantics. If a service module that is designated as *requisite* fails, then the PAM framework immediately returns an error to the application, and all subsequent service modules in the stack are ignored. If a prior *required* service module has failed, then that error is returned. If no prior *required* service module failed, then the error from the failed *requisite* service module is returned.

If a service module that is designated as *sufficient* succeeds, then the PAM framework immediately returns success to the application, and all subsequent services modules in the stack, even *requisite* and *required* ones, are ignored, given that all prior *requisite* and *required* modules have also succeeded. If a prior *required* module has failed, then the error value from that module is returned.

If any entry in `pam.conf` is incorrect, or if a module does not exist or cannot be opened, then all PAM services will fail and users will not be permitted access to the system. An error will be logged through `syslog`(3C) at the `LOG_CRIT` level. To fix incorrect entries in `pam.conf`, a system administrator may boot the system in maintenance mode (single user) to edit the file. Below is a sample configuration file that stacks the `su`, `login`, and `rlogin` services.

```
su     auth  requisite  /usr/lib/security/$ISA/pam_inhouse.so.1
su     auth  required   /usr/lib/security/$ISA/pam_unix.so.1         debug
login  auth  required   /usr/lib/security/$ISA/pam_unix.so.1         debug
login  auth  optional   /usr/lib/security/$ISA/pam_inhouse.so.1
rlogin auth  sufficient /usr/lib/security/$ISA/pam_rhosts_auth.so.1
rlogin auth  required   /usr/lib/security/$ISA/pam_unix.so.1
```

In the case of `su`, the user is authenticated by the Inhouse and UNIX authentication modules. Because the Inhouse and UNIX authentication modules are *requisite* and *required*, respectively, an error is returned back to the application if either module fails. In addition, if the *requisite* authentication (Inhouse authentication) fails, the UNIX authentication module is never invoked, and the error is returned immediately back to the application.

In the case of `login`, the *required* keyword for *control_flag* requires that the user be allowed to login only if the user is authenticated by the UNIX service module. If UNIX authentication fails, control continues to proceed down the stack, and the Inhouse authentication module is invoked. Inhouse authentication is optional by virtue of the *optional* keyword in the *control_flag* field. The user can still log in even if Inhouse authentication fails, assuming the UNIX authentication succeeded.

In the case of `rlogin`, the *sufficient* keyword for *control_flag* specifies that if the *rhosts* authentication check succeeds, then PAM should return success to `rlogin` and `rlogin` should not prompt the user for a password. The UNIX authentication module, which is the next module in the stack, will only be invoked if the *rhosts* check fails. This gives the system administrator the flexibility to determine if *rhosts* alone is sufficient enough to authenticate a remote user.

Some modules may return PAM_IGNORE in certain situations. In these cases the PAM framework ignores the entire entry in `pam.conf` regardless of whether or not it is *requisite*, *required*, *optional* or *sufficient*.

Utilities and Files

A following is a list of the utilities that are known to use PAM: include: `login`, `passwd`, `su`, `rlogind`, `rshd`, `telnetd`, `ftpd`, `rpc.rexd`, `uucpd`, `init`, `sac`, and `ttymon`.

The utility `dtlogin` also uses PAM. Note however that `dtlogin` is the login service utility for the Common Desktop Environment (CDE).

The PAM configuration file does not dictate either the name or the location of the service specific modules. The convention, however, is the following:

`/usr/lib/security/$ISA/`*pam_module_name*`.so.x`
Implements various function of specific authentication services.

`/etc/pam.conf`
Configuration file.

`/usr/lib/$ISA/libpam.so.1`
Implements the PAM framework library.

EXAMPLES

EXAMPLE 1 A sample pam.conf configuration file.

The following is a sample pam.conf configuration file. Lines that begin with the # symbol are treated as comments, and therefore ignored.

```
#
# PAM configuration

#
# Authentication management for login service is stacked.
# Both UNIX and inhouse authentication functions are invoked.
login   auth required /usr/lib/security/$ISA/pam_unix.so.1
login   auth required /usr/lib/security/$ISA/pam_inhouse.so.1 try_first_pass
dtlogin auth required /usr/lib/security/$ISA/pam_unix.so.1
dtlogin auth required /usr/lib/security/$ISA/pam_inhouse.so.1 try_first_pass
#
# Authentication management for rlogin service is stacked.
# If the rhost check succeeds, do not continue
rlogin   auth   sufficient /usr/lib/security/$ISA/pam_rhosts_auth.so.1
rlogin   auth   required   /usr/lib/security/$ISA/pam_unix.so.1
#
# Other services use UNIX authentication
other   auth   required   /usr/lib/security/$ISA/pam_unix.so.1
#
# Account management for login service is stacked.
# UNIX account management is required
# Inhouse account management is optional
login     account   required  /usr/lib/security/$ISA/pam_unix.so.1
login     account   optional  /usr/lib/security/$ISA/pam_inhouse.so.1
dtlogin   account   required  /usr/lib/security/$ISA/pam_unix.so.1
dtlogin   account   optional  /usr/lib/security/$ISA/pam_inhouse.so.1
other     account   required  /usr/lib/security/$ISA/pam_unix.so.1
#
# Session management
other   session   required   /usr/lib/security/$ISA/pam_unix.so.1
#
# Password management
other   password   required   /usr/lib/security/$ISA/pam_unix.so.1
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
MT Level	MT-Safe with exceptions

SEE ALSO

login(1), passwd(1), in.ftpd(1M), in.rlogind(1M), in.rshd(1M), in.telnetd(1M), in.uucpd(1M), init(1M), rpc.rexd(1M), sac(1M), su(1M), ttymon(1M), pam(3PAM), syslog(3C), libpam(3LIB), attributes(5), pam_unix(5)

NOTES

The interfaces in libpam() are MT-Safe only if each thread within the multi-threaded application uses its own PAM handle.

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. It is recommended that this field conform to the checks performed by `pwck`(1M).

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.) An "&" (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 maximum value of the *uid* and *gid* fields is `2147483647`. To maximize interoperability and compatibility, administrators are recommended to assign users a range of UIDs and GIDs below `60000` where possible.

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

EXAMPLE 1 A sample `passwd` file.

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` is:

```
passwd:  compat
```

then 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), `newgrp`(1), `nispasswd`(1), `passwd`(1), `sh`(1), `sort`(1), `chown`(1M), `domainname`(1M), `getent`(1M), `in.ftpd`(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), environ(5), unistd(3HEAD)

System Administration Guide, Volume 1

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

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* "*driver binding name*"

where

physical name	is the absolute physical pathname of a device. This pathname must be enclosed in double quotes.
instance number	is a decimal or hexadecimal number.
driver binding name	is the name used to determine the driver for the device. This name may be a driver alias or a driver name. The driver binding name must be enclosed in double quotes.

EXAMPLES

EXAMPLE 1 Sample `path_to_inst` Entries

Here are some sample `path_to_inst` entries:

```
"/iommu@f,e0000000" 0 "iommu"
"/iommu@f,e0000000/sbus@f,e0001000" 0 "sbus"
"/iommu@f,e0000000/sbus@f,e0001000/sbusmem@e,0" 14 "sbusmem"
"/iommu@f,e0000000/sbus@f,e0001000/sbusmem@f,0" 15 "sbusmem"
```

```
"/iommu@f,e0000000/sbus@f,e0001000/ledma@f,400010" 0 "ledma"
"/obio/serial@0,100000" 0 "zs"
"/SUNW,sx@f,80000000" 0 "SUNW,sx"
```

FILES

/etc/path_to_inst

SEE ALSO

add_drv(1M), boot(1M), drvconfig(1M), mknod(1M)

WARNINGS

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

pci – configuration files for PCI device drivers

DESCRIPTION

The Peripheral Component Interconnect (PCI) bus is a little endian bus. PCI devices are *self-identifying* — that is to say the PCI device provides configuration parameters to the system which allows the system to identify the device and its driver. The configuration parameters are represented in the form of name-value pairs that can be retrieved using the DDI property interfaces. See `ddi_prop_lookup`(9F) for details.

The PCI bus properties are derived from PCI Configuration Space, or supplied by the Fcode PROM if it exists. Therefore, driver configuration files are not necessary for these devices.

However, on some occasions, drivers for PCI devices may use driver configuration files to provide driver private properties. This can be done through global property mechanism. See `driver.conf`(4) for further details. Driver configuration files can also be used to augment or override properties for a specific instance of a driver.

All bus drivers of class `pci` recognize the following properties:

`reg` An arbitrary length array where each element of the array consists of a 5-tuple of 32-bit values. Each array element describes a logically contiguous mappable resource on the PCI bus.

The first 3 values in the 5-tuple describe the PCI address of the mappable resource. The first tuple contains the following information:

Bits 0 - 7	8-bit Register number
Bits 8 - 10	3-bit Function number
Bits 11 - 15	5-bit Device number
Bits 16 - 23	8-bit Bus number
Bits 24 - 25	2-bit Address Space type identifier

The Address Space type identifier may be interpreted as follows:

0x0	Configuration Space
0x1	I/O Space

0x2	32-bit Memory Space address
0x3	64-bit Memory Space address

The Bus number is a unique identifying number assigned to each PCI bus within a PCI domain.

The Device number is a unique identifying number assigned to each PCI device on a PCI bus. Note that a Device number is only unique within the set of Device numbers for a particular bus.

Each PCI device can have 1 to 8 logically independent functions, each with its own independent set of configuration registers. Each function on a device is assigned a Function number. For a PCI device with only one function, the Function number must be `0`.

The Register number field selects a particular register within the set of configuration registers corresponding to the selected function.

The second and third values in the `reg` property 5-tuple specify the 64-bit address of the mappable resource within the PCI address domain. The second 32-bit tuple corresponds to the high order 4 bytes of the 64-bit address. The third 32-bit tuple corresponds to the low order bytes.

The fourth and fifth 32-bit values in the 5-tuple `reg` property specify the size of the mappable resource. The size is a 64-bit value where the fourth tuple corresponds to the high order bytes of the 64-bit size and the fifth corresponds to the low order.

The driver can refer to the elements of this array by index, and construct kernel mappings to these addresses using `ddi_regs_map_setup`(9F). The index into the array is passed as the *rnumber* argument of `ddi_regs_map_setup`(9F).

At a high-level interrupt context, you can use the `ddi_get*` and `ddi_put*` family of functions to access I/O and memory space. However, access to configuration space is not allowed when running at a high-interrupt level.

`interrupts` This property consists of a single integer element array. Valid interrupt property values are `1`, `2`, `3`, and `4`. This value is derived directly from the contents of the device's Configuration Interrupt Pin register.

A driver should use an index value of `0` when registering its interrupt handler with `ddi_add_intr`(9F).

All PCI devices support the reg property. The Device number and Function number as derived from the `reg` property are used to construct the address part of the device name under `/devices`.

Only devices that generate interrupts support an `interrupts` property.

Occasionally it may be necessary to override or augment the configuration information supplied by a PCI device. This can be achieved by writing a driver configuration file that describes a prototype device node specification containing the additional properties required.

For the system to merge the prototype node specification into an actual device node, certain conditions must be met. First, the name property must be identical. Second, the parent property must identify the PCI bus. Third, the unit-address property must identify the card. The format of the unit-address property is

```
DD[,F]
```

where DD is the device number and F is the function number. If the function number is 0, only DD is specified.

EXAMPLES

EXAMPLE 1 A sample configuration file.

An example configuration file called `ACME,scsi-hba.conf` for a PCI driver called `ACME,scsi-hba` follows:

```
#
# Copyright (c) 1995, ACME SCSI Host Bus Adaptor
# ident   "@(#)ACME,scsi-hba.conf  1.1  96/02/04"
name="ACME,scsi-hba" parent="/pci@1,0/pci@1f,4000"
   unit-address="3" scsi-initiator-id=6;
hba-advanced-mode="on";
hba-dma-speed=10;
```

In this example, we provide a property scsi-initiator-id to specify the SCSI bus initiator id that the adapter should use, for just one particular instance of adapter installed in the machine. We use the name property to identify the driver and the parent property to identify the particular bus the card is plugged into. This example uses the parent's full path name to identify the bus. The unit-address property identifies the card itself, with device number of 3 and function number of 0.

Two global driver properties are also created: `hba-advanced-mode` (which has the string value `on`) and `hba-dma-speed` (which has the value `10` M bit/s). These properties apply to all device nodes of the `ACME,scsi-hba`. The following is an example configuration file called `ACME,foo.conf` for a PCI driver called `ACME,foo`;

```
#
# Copyright (c) 1996, ACME Foo driver
# ident   "@(#)ACME,foo.conf  1.1  95/11/14"
name="ACME,foo" class="pci" unit-address="3,1"
   debug-mode=12;
```

In this example, we provide a property `debug-mode` for all instances of the `ACME,foo` driver with parents of class `pci` and device and function numbers of `3` and `1`, respectively.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC, IA

SEE ALSO

`driver.conf`(4), `attributes`(5), `ddi_add_intr`(9F), `ddi_prop_lookup`(9F), `ddi_regs_map_setup`(9F)

Writing Device Drivers

IEEE 1275 PCI Bus Binding

NAME

pcmcia – PCMCIA nexus driver

DESCRIPTION

The PCMCIA nexus driver supports PCMCIA card client device drivers. There are no user-configurable options for this driver.

FILES

`/kernel/misc/pcmcia` `pcmcia` driver

SEE ALSO

`pcmciad`(1M)

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. 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. It is customary to make the first four letters unique to your company, such as the company's stock symbol.

NAME* Text that specifies the package name (maximum length of 256 ASCII characters). Use the NAME parameter as the foundation for describing the functionality and purpose of the package; spell out any acronyms and avoid internal product / project code names. The DESC parameter can then be used to expand the descriptive information. Use the NAME parameter to state as specifically as possible the use of the package, why a user would need to load it, and so on.

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.

Solaris 2 and Solaris 7's installation software meaningfully uses only one architecture token of the form:

<*instruction_set_architecture*>[.<*platform_group*>]

where *platform_group* is intended only for Solaris installation packages. Third party application software should restrict itself to ARCH values from the following Solaris-supported instruction set architectures (uname -p): sparc, i386, and

	ppc. Examples of Solaris' platform groups (uname -m) are sun4u, sun4d, and sun4m for the SPARC® instruction set and i86pc for the i386 instruction set. See uname(1) and isalist(1) for more details.
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. Current Solaris and Solaris-compatible software practice is to assign this parameter monotonically increasing Dewey decimal values of the form: <*major_revision*>.<*minor_revision*>[.<*micro_revision*>] where all the revision fields are integers. The versioning fields can be extended to an arbitrary string of numbers in Dewey-decimal format, if necessary.
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). This parameter value is used to provide the installer with a description of what the package contains and should build on the description provided in the NAME parameter. Try to make the two parameters work together so that a pkginfo -l will provide a fairly comprehensive textual description of the package.
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" allows run states of S, s or 1). Solaris 2 and Solaris 7 support the run levels s, S, 0, 1, 2, 3, 5, and 6. Applicable run levels for this parameter are s, S, 1, 2, and 3. See init(1M) for details.
RSTATES	A list of allowable run states for package removal (for example, "S s 1" allows run states of S, s or 1). Solaris 2 and Solaris 7 support the run levels s, S, 0, 1, 2, 3, 5, and 6. Applicable run levels for this parameter are s, S, 1, 2, and 3 See init(1M) for details.
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(1) command (see limit(1)), 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. In order to support multiple instances of packages (for example, packages that differ in their ARCH or VERSION parameter value), the value of this parameter must be high enough to allow for all instances of a given package, including multiple versions coexisting on a software server.
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 "*YYYYMMDDHHMM*" (year, month, date, hour, minutes).

`INTONLY` Indicates that the package should only be installed interactively when set to any non-null value.

`SUNW_PRODNAME` Solaris 2 and Solaris 7-only parameter indicating the name of the product this package is a part of or comprises (maximum length of 256 ASCII characters). A few examples of currently used `SUNW_PRODNAME` values are: `"SunOS"`, `"OpenWindows"`, and `"Common Desktop Environment"`.

`SUNW_PRODVERS` Solaris 2 and Solaris 7-only parameter indicating the version or release of the product described in `SUNW_PRODNAME` (maximum length of 256 ASCII characters). For example, where `SUNW_PRODNAME="`*SunOS*`"`, and the Solaris 2.x Beta release, this string could be `"5.x BETA"`, while for the Solaris 2.x FCS release, the string would be `"5.x"`. For Solaris 7, the string is `"5.7"`. If the `SUNW_PRODNAME` parameter is `NULL,` so should be the `SUNW_PRODVERS` parameter.

`SUNW_PKGVERS` Solaris 2 and Solaris 7–only parameter indicating of version of the Solaris 2 or Solaris 7 package interface. It is used to indicate the version of the Solaris 2 or Solaris 7-specific software packaging interfaces.

`SUNW_PKGVERS="`*<sunw_package_version>*`"`

where <*unw_package_version*> has the form *x.y[.z]* and *x*, *y*, and *z* are integers. For packages built for this release and previous releases, use `SUNW_PKGVERS`="*1.0*".

`SUNW_PKGTYPE` Solaris 2 and Solaris 7-only parameter for Sun internal use only. Required for packages part of the Solaris 2 and Solaris 7 releases which install into the `/`, `/usr`, `/usr/kvm`, and `/usr/openwin` file systems. The Solaris 2 and Solaris 7 installation software must know which packages are part of which file system to properly install a server/client configuration. The currently allowable values for this parameter are `root`, `usr`, `kvm`, and `ow`. If no `SUNW_PKGTYPE` parameter is present, the package is assumed to be of `BASEDIR=` */opt*. `SUNW_PKGTYPE` is optional only for packages which install into the `/opt` name space as is the case for the majority of Solaris 2 and Solaris

7-compatible add-on software. See the SUNW_PKGTYPE parameter in packagetoc(4) for further information.

SUNW_ISA Solaris 2 and Solaris 7-only optional parameter that indicates a software package contains 64–bit objects if it is set to sparc9. If this parameter is not set, the default *ISA* (instruction set architecture) is set to the value of the ARCH parameter.

SUNW_LOC Solaris 2 and Solaris 7-only optional parameter used to indicate a software package containing localization files for a given product or application. The parameter value is a comma-separated list of locales supported by a package. It is only used for packages containing localization files, typically the message catalogues. The allowable values for this string field are those found in the table of Standard Locale Names located in the *International Language Environments Guide*.

```
SUNW_LOC="<locale_name>,<locale_name>,..,<locale_name>"
```

where

```
<locale_name>::= <language>[_<territory>][.<codeset>]
<language>::= the set of names from ISO 639
<territory>::= the set of territories specified
in ISO 3166
<codeset>::= is a string corresponding to the coded
character set
```

Since a value of C specifies the traditional UNIX system behavior (American English, en_US), packages belonging to the C locale are viewed as non-localized packages, and thus must not have SUNW_LOC and SUNW_PKGLIST included in their pkginfo file. See also the SUNW_LOC parameter in packagetoc(4) and setlocale(3C) for more information. This keyword is not recognized by the add-on software utility Software Manager.

SUNW_PKGLIST Solaris 2 and Solaris 7-only optional parameter used to associate a localization package to the package(s) from which it is derived. It is required whenever the SUNW_LOC parameter is defined. This parameter value is an comma-separated list of package abbreviations of the form:

```
SUNW_PKGLIST="pkg1[:version],pkg2[:version],..."
```

where *version* (if specified) should match the version string in the base package specified (see VERSION parameter in this manual page). When in use, SUNW_PKGLIST helps determine the order of package installation. The packages listed in the parameter will be installed before the localization package in question is installed. When left blank, SUNW_PKGLIST=" ", the package is assumed to be required for the locale to function correctly. See the SUNW_PKGLIST parameter in packagetoc(4) for more information. This keyword is not recognized by the add-on software utility Software Manager.

EXAMPLES

EXAMPLE 1 A sample pkginfo file.

Here is a sample pkginfo file:

```
SUNW_PRODNAME="SunOS"
SUNW_PRODVERS="5.5"
SUNW_PKGTYPE="usr"
PKG="SUNWesu"
NAME="Extended System Utilities"
VERSION="11.5.1"
ARCH="sparc"
VENDOR="Sun Microsystems, Inc."
HOTLINE="Please contact your local service provider"
EMAIL=""
VSTOCK="0122c3f5566"
CATEGORY="system"
ISTATES="S 2"
RSTATES="S 2"
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWcsu
Interface Stability	See entries below
PKG value	Evolving
VERSION value	Evolving
NAME value	Evolving
DESC value	Evolving
ARCH value	Evolving
CATEGORY value	Evolving
BASEDIR value	Evolving

ATTRIBUTE TYPE	ATTRIBUTE VALUE
ISTATES value	Evolving
RSTATES value	Evolving
MAXINST value	Evolving
SUNW_PRODNAME	Evolving
SUNW_PRODVERS	Evolving
SUNW_PKGVERS	Evolving
SUNW_PKGTYPE	Unstable
SUNW_LOC	Evolving
SUNW_PKGLIST	Evolving

SEE ALSO

`isalist`(1), `limit`(1), `pkgmk`(1), `uname`(1), `init`(1M), `setlocale`(3C), `clustertoc`(4), `order`(4), `packagetoc`(4), `pkgmap`(4), `attributes`(5)

Application Packaging Developer's Guide

International Language Environments Guide

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.

Trailing white space after any parameter value is ignored. For example, `VENDOR="Sun Microsystems, Inc."` is the same as `VENDOR="Sun Microsystems, Inc.  "`.

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(4) file.

Each entry in pkgmap describes a single "deliverable object file." A deliverable object file includes shell scripts, executable objects, data files, directories, and so forth. 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:

- b — block special device
- c — character special device
- d — directory
- e — a file to be edited upon installation or removal (may be shared by several packages)
- f — a standard executable or data file
- i — installation script or information file
- l — linked file
- p — named pipe
- s — symbolic link
- v — volatile file (one whose contents are expected to change, like a log file)
- x — an exclusive directory accessible only by this package

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 of the form $*variable* that support install-time configuration of the file. *variable* may be embedded in the pathname structure. (See `prototype`(4) for definitions of variable specifications.)

  Do not use the following reserved words in *pathname*, since they are applied by `pkgadd`(1M) 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.

  The mode can contain a variable specification. (See `prototype`(4) for definitions of variable specifications.)

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.

  The owner can contain a variable specification. (See `prototype`(4) for definitions of variable specifications.)

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.

The group can contain a variable specification. (See `prototype`(4) for definitions of variable specifications.)

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` file must have one line that provides information about the number of parts, maximum size of parts that make up the package, and, optionally, the size of the package after compression (where size is given in 512-byte blocks). This line is in the following format:

`:` *number_of_parts maximum_part_size compressed_pkg_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

EXAMPLE 1 A sample 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), pkgadd(1M), stat(2), pkginfo(4), prototype(4)

Application Packaging Developer's Guide

NOTES

The pkgmap file may contain only one entry per unique pathname.

NAME

platform – directory of files specifying supported platforms

SYNOPSIS

`.platform`

DESCRIPTION

The Solaris 2.5 release includes the `.platform` directory, a new directory on the Solaris CD image. This directory contains files (created by SunSoft and Solaris OEMs) that define platform support. These files are generically referred to as *platform definition files*. They provide a means to map different platform types into a platform group.

Platform definition files in the .platform directory are used by the installation software to ensure that software appropriate for the architecture of the system will be installed.

SunSoft provides a platform definition file named `.platform/Solaris`. This file is the only one that can define platform groups to which other platform definition files can refer. For example, an OEM platform definition file can refer to any platform group specified in the Solaris platform definition file.

Other platform definition files are delivered by OEMs. To avoid name conflicts, OEMs will name their platform definition file with an OEM-unique string. OEM's should use whatever string they use to make their package names unique. This unique string is often the OEM's stock symbol.

Comments are allowed in a platform definition file. A "#" begins a comment and can be placed anywhere on a line.

Platform definition files are composed of keyword-value pairs, and there are two kinds of stanzas in the file: platform group definitions and platform identifications.

- Platform group definitions:

  The keywords in a platform group definition stanza are:

`PLATFORM_GROUP` The `PLATFORM_GROUP` keyword *must* be the first keyword in the platform group definition stanza. The value assigned to this keyword is the name of the platform group, for example:

PLATFORM_GROUP=sun4c

The `PLATFORM_GROUP` name is an arbitrary name assigned to a group of platforms. However, `PLATFORM_GROUP` typically equals the output of the uname -m command. `PLATFORM_GROUP` value cannot have white space and is limited to 256 ASCII characters.

`INST_ARCH` The instruction set architecture of all platforms in the platform group, for example:

INST_ARCH=sparc

The `INST_ARCH` keyword value must be the value returned by the uname -p command on all platforms in the platform group.

- Platform identifications:

The keywords in a platform identification stanza are:

`PLATFORM_NAME`
: The `PLATFORM_NAME` keyword *must* be the first keyword in the platform identification stanza. The `PLATFORM_NAME` is the name assigned to the platform, for example:

PLATFORM_NAME=SUNW,SPARCstation-5

Typically, this name is the same as the value returned by the uname -i command on the machine, but it need not be the same.

The `PLATFORM_NAME` value cannot have white space and is limited to 256 ASCII characters. If it contains parentheses, it must contain only balanced parentheses. For example. the string "foo(bar)foo" is a valid value for this keyword, but "foo(bar" is not.

The other keywords in the platform identification stanza can be in any order, as long as the `PLATFORM_NAME` keyword is first.

`PLATFORM_ID`
: The value returned by the uname -i command on the machine, for example:

PLATFORM_ID=SUNW,SPARCstation-5

`MACHINE_TYPE`
: The value returned by the uname -m command on the machine, for example:

MACHINE_TYPE=sun4c

`IN_PLATFORM_GROUP`
: The platform group of which the platform is a member, for example:

IN_PLATFORM_GROUP=sun4c

The platform group name must be specified in the same file as the platform identification stanza or in the platform definition file with the name `.platform/Solaris` .

The `IN_PLATFORM_GROUP` keyword is optional. A platform doesn’t have to belong to a platform group. If a platform isn't explicitly assigned to a platform group, it essentially forms its own platform group, where the platform group name is the PLATFORM_NAME value. The IN_PLATFORM_GROUP value typically equals the output of the uname -m command. IN_PLATFORM_GROUP value cannot have white space and is limited to 256 ASCII characters.

INST_ARCH
: The instruction set architecture of the platform, for example:

  INST_ARCH=sparc

  This field is only required if the platform does not belong to a platform group. The INST_ARCH keyword value must be the value returned by the uname -p command on all platforms in the platform group.

COMPATIBILITY

The installation program will remain compatible with the old Solaris CD format. If a Solaris CD image does not contain any platform definition files, the installation and upgrade programs will select the packages to be installed based on machine type (i.e., the value returned by the uname -m command).

EXAMPLES

EXAMPLE 1 The following example shows platform group definitions from the .platform / Solaris platform definition file.

```
#
PLATFORM_GROUP=sun4c
INST_ARCH=sparc
#
PLATFORM_GROUP=sun4d
INST_ARCH=sparc
#
PLATFORM_GROUP=sun4m
INST_ARCH=sparc
#
PLATFORM_GROUP=sun4u
INST_ARCH=sparc
```

EXAMPLE 2 The following example shows platform identification stanzas, which define systems that belong in a platform group, from the .platform / Solaris platform definition file.

```
#
PLATFORM_NAME=SUNW,Sun_4_20
PLATFORM_ID=SUNW,Sun_4_20
IN_PLATFORM_GROUP=sun4c
PLATFORM_NAME=SUNW,Sun_4_25
PLATFORM_ID=SUNW,Sun_4_25
```

```
IN_PLATFORM_GROUP=sun4c
#
PLATFORM_NAME=SUNW,SPARCstation-5
PLATFORM_ID=SUNW,SPARCstation-5
IN_PLATFORM_GROUP=sun4m
#
PLATFORM_NAME=SUNW,SPARCstation-10
PLATFORM_ID=SUNW,SPARCstation-10
IN_PLATFORM_GROUP=sun4m
```

FILES

The `.platform` directory must reside as /*cd_image*/`Solaris_`*vers*`/.platform`, where

cd_image	Is the path to the mounted Solaris CD (/cdrom/cdrom0/s0 by default) or the path to a copy of the Solaris CD on a disk.
Solaris_*vers*	Is the version of Solaris: e.g., Solaris_2.5.

NOTES

Typically, a platform identification stanza contains either a `PLATFORM_ID` or a `MACHINE_TYPE` stanza, but *not* both.

If both are specified, both must match for a platform to be identified as this platform type. Each platform identification stanza must contain either a `PLATFORM_ID` value or a `MACHINE_TYPE` value. If a platform matches two different platform identification stanzas—one which matched on the value of `PLATFORM_ID` and one which matched on the value of `MACHINE_TYPE` , the one that matched on `PLATFORM_ID` will take precedence.

The `.platform` directory is part of the Solaris CD image, whether that be the Solaris CD or a copy of the Solaris CD on a system's hard disk.

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

policy.conf – configuration file for security policy

SYNOPSIS

etc / security / policy.conf

DESCRIPTION

The `policy.conf` file provides the security policy configuration for user-level attributes. Each entry consists of a of a key / value pair in the form:

key=value

The following key is defined:

AUTHS_GRANTED	Specifies the default set of authorizations granted to all users. This entry is interpreted by `chkauthattr`(3SECDB). The value is one or more comma-separated authorizations defined in `auth_attr`(4).

The key / value pair must appear on a single line, and the key must start the line. Lines starting with # are taken as comments and ignored. Option name comparisons are case-insensitive.

EXAMPLES

EXAMPLE 1 Defining a key / value pair

```
AUTHS_GRANTED=com.sun.date
```

FILES

`/etc/user_attr`	Defines extended user attributes.
`/etc/security/auth_attr`	Defines authorizations.
`/etc/security/policy.conf`	Defines policy for the system.

SEE ALSO

`pfexec`(1), `chkauthattr`(3SECDB), `auth_attr`(4), `user_attr`(4)

NAME

power.conf – Power Management configuration information file

SYNOPSIS

`/etc/power.conf`

DESCRIPTION

The `power.conf` file is used by the Power Management configuration program `pmconfig`(1M) to initialize the settings for Power Management. If you make changes to this file, you must run `pmconfig`(1M) manually for the changes to take effect.

The `dtpower`(1M) GUI allows the configuration of a subset of parameters allowed by this file. For ease-of-use, it is recommended that you use `dtpower`(1M) to configure the parameters.

Power Management addresses two specific management scenarios: management of individual devices and management of the whole system. An individual device is power managed if a device supports multiple power levels and if the device driver uses Power Management interfaces provided by the kernel to save device power when the device is idle. If the driver uses the original Power Management interfaces, the device is controlled by the entries described in the DEVICE POWER MANAGEMENT section of this manual page. If the device driver uses new automatic device Power Management interfaces, the device is controlled by the entries described in the AUTOMATIC DEVICE POWER MANAGEMENT section of this manual page.

To determine if the device driver supports original Power Management interfaces, contact the device vendor. To find out if the device driver supports the new automatic device Power Management interfaces, look for "pm-components" property (`pm-components`(9F)) under the device name from the output of `prtconf -v` command (`prtconf`(1M).)

The original Power Management interfaces and the corresponding device Power Management entries in `power.conf` file that were supported in Solaris 7 and earlier releases are now obsolete. Support for them will be removed in a future release.

All entries in the `power.conf` file are processed in the order displayed in the file.

Device Power Management

Device Power Management entries are now obsolete and support for them will be removed in a future release. If a device supports original Power Management interfaces, it needs to be explicitly configured for Power Management using an entry of the form shown below. A device will not be power managed if there is no entry for the device. Be sure you fully understand the Power Management framework before you attempt to modify device Power Management entries.

Device Power Management entries consist of line-by-line listings of the devices to be configured. Each line is of the form:

device_name threshold ...dependent_upon...

The fields must be in the order shown above. Each line must contain a *device_name* field and a *threshold* field; it may also contain a *dependent_upon* field. Fields and sub-fields are separated by white space (tabs or spaces). A line may be more than 80 characters. If a newline character is preceded by a backslash (\) it will be treated as white space. Comment lines must begin with a hash character (#).

The *device_name* field specifies the device to be configured. *device_name* is either a pathname specifying the device special file or a relative pathname containing the name of the device special file. (For the latter format, you can avoid using the full pathname by omitting the pathname component that specifies the parent devices. This includes the leading '/'.) Using the relative pathname format, the first device found with a full pathname containing *device_name* as its tail is matched. In either case, the leading `/devices` component of the pathname does not need to be specified.

The *threshold* field is used to configure the power manageable components of a device. These components represent entities within a device that may be power-managed separately. This field may contain as many integer values as the device has components. Each *threshold* time specifies the idle time in seconds before the respective component may be powered down. If there are fewer component *threshold* times than device components, the remaining components are not power managed. Use a value of `−1` to explicitly disable power-down for a component. At least one component *threshold* must be specified per device (in the file).

The *dependent_upon* field contains a list of devices that must be idle and powered-down before the dependent device in *device_name* field can be powered down. A device must previously have been configured before it can be used in *dependent_upon* list. This field should only list logical dependents for this device. A logical dependent is a device that is not physically connected to the power managed device (for example, the display and the keyboard). Physical dependents are automatically considered and do not need to be included.

A device Power Management entry is only effective if there is no user process controlling the device directly. For example, X Window systems directly control framebuffers and entries in this file are effective only when X Windows are not running.

Automatic Device Power Management

Devices whose drivers use the new automatic device Power Management interfaces (as evident by existence of `pm-components`(9) property) are automatically power managed if enabled by the `autopm` entry described below.

When a component has been idle at a given power level for its *threshold* time, the power level of the component will be reduced to the next lower power level of that component (if any). For devices which implement multiple components, each component is power-managed independently.

Default thresholds for components of automatically power managed devices are computed by the Power Management framework based on the system idleness *threshold*. By default, all components of the device are powered off if they have all been idle for the system's idleness *threshold*. The default system idleness *threshold* is determined by the applicable United States Environmental Protection Agency's (EPA) *Energy Star Memorandum of Understanding*. See the NOTES section of this manual page for more information.

To set the system idleness *threshold*, use one of the following entries:

`system-threshold` *threshold*

`system-threshold always-on`

where *threshold* is the value of the system idleness threshold in hours, minutes or seconds as indicated by a trailing `h`, `m` or `s` (defaulting to seconds if only a number is given). If `always-on` is specified, then by default, all devices will be left at full power.

To override the default device component thresholds assigned by the Power Management framework, a `device-thresholds` entry may be used. A `device-thresholds` entry sets thresholds for a specific automatically power-managed device or disables automatic Power Management for the specific device.

A `device-thresholds` entry has the form:

`device-thresholds` *phys_path* *(threshold ...) ...*

or

`device-thresholds` *phys_path* *threshold*

or

`device-thresholds` *phys_path* `always-on`

where *phys_path* specifies the physical path (`libdevinfo`(3)) of a specific device. For example, `/pci@8,600000/scsi@4/ssd@w210000203700c3ee,0` specifies the physical path of a disk. A symbolic link into the `/devices` tree (for example `/dev/dsk/c1t1d0s0`) is also accepted. The thresholds apply (or keeping the device always on applies) to the specific device only.

In the first form above, each *threshold* value represents the number of hours, minutes or seconds (depending on a trailing `h`, `m` or `s` with a default to seconds) to spend idle at the corresponding power level before power will be reduced to the next lower level of that component. Parentheses are used to group thresholds per component, with the first (leftmost) group being applied to component `0`, the next to component `1`, etc. Within a group, the last (rightmost) number represents the time to be idle in the highest power level of the component before going to the next-to-highest level, while the first (leftmost) number represents the time to be idle in the next-to-lowest power level before going to the lowest power level.

If the number of groups does not match the number of components exported by the device (via `pm-components`(9) property), or the number of thresholds in a group is not one less than the number of power levels the corresponding component supports, then an error message will be printed and the entry will be ignored.

For example, assume a device called *xfb* exports the components *Frame Buffer* and *Monitor*. Component *Frame Buffer* has two power levels: `Off` and `On`. Component *Monitor* has four power levels: `Off`, `Suspend, Standby`, and `On`.

The following `device-thresholds` entry:

`device-thresholds` */pci@f0000/xfb@0 (0) (3m 5m 15m)*

would set the *threshold* time for the *Monitor* component of the specific *xfb* card to go from `On` to `Standby in` 15 minutes, the *threshold* for *Monitor* to go from `Standby` to `Suspend` in 5 minutes, and the *threshold* for *Monitor* to go from `Suspend` to `Off` in 3 minutes. The threshold for *Frame Buffer* to go from `On` to `Off` will be 0 seconds.

In the second form above, where a single *threshold* value is specified without parentheses, the *threshold* value represents a maximum overall time within which the entire device should be powered down if it is idle. Because the system does not know about any internal dependencies there may be among a device's components, the device may actually be powered down sooner than the specified *threshold*, but will not take longer than the specified *threshold*, provided that all device components are idle.

In the third form above, all components of the device are left at full power.

Device Power Management entries are only effective if there is no user process controlling the device directly. For example, X Window systems directly control frame buffers and the entries in this file are effective only when X Windows are not running.

Dependencies among devices may also be defined. A device depends upon another if none of its components may have their power levels reduced unless all components of the other device are powered off. A dependency may be indicated by an entry of the form:

`device-dependency` *dependent_phys_path phys_path [phys_path ...]*

where *dependent_phys_path* is the path name (as above) of the device that is kept up by the others, and the *phys_path* entries specify the devices that keep it up. A symbolic link into the `/devices` tree (such as `/dev/fb`) is also accepted. This entry is needed only for logical dependents for the device. A logical dependent is a device that is not physically connected to the power managed device (for example, the display and the keyboard). Physical dependents are automatically considered and need not be included.

An `autopm` entry may be used to enable or disable automatic device Power Management on a system-wide basis. The format of the `autopm` entry is:

`autopm` *behavior*

Acceptable behavior values and their meanings are:

`default` The behavior of the system will depend upon its model. Desktop models that fall under the United States Environmental Protection Agency's *Energy Star Memorandum of Understanding #3* will have automatic device Power Management enabled, and all others will not. See the NOTES section of this manual page for more information.

`enable` Automatic device Power Management will be started when this entry is encountered.

`disable` Automatic device Power Management will be stopped when this entry is encountered.

System Power Management

The system Power Management entries control power management of the entire system using the suspend-resume feature. When the system is suspended, the complete current state is saved on the disk before power is removed. On reboot, the system automatically starts a resume operation and the system is restored to the state it was in prior to suspend.

The system can be configured to do an automatic shutdown (autoshutdown) using the suspend-resume feature by an entry of the following form:

`autoshutdown` *idle_time start_time finish_time behavior*

idle_time specifies the time in minutes that system must have been idle before it will be automatically shutdown. System idleness is determined by the inactivity of the system and can be configured as discussed below.

start_time and *finish_time* (each in `hh:mm`) specify the time period during which the system may be automatically shutdown. These times are measured from the start of the day (12:00 a.m.). If the *finish_time* is less than or equal to the *start_time*, the period span from midnight to the *finish_time* and from the *start_time* to the following midnight. To specify continuous operation, the *finish_time* may be set equal to the *start_time*.

Acceptable *behavior* values and their meanings are:

`shutdown` The system will be shut down automatically when it has been idle for the number of minutes specified in the *idle_time* value and the time of day falls between the *start_time* and *finish_time* values.

`noshutdown` The system is never shut down automatically.

`autowakeup` If the hardware has the capability to do `autowakeup,` the system is shut down as if the value were `shutdown` and the system will be restarted automatically the next time the time of day equals *finish_time*.

`default` The behavior of the system will depend upon its model. Desktop models that fall under the United States Enviromental Protection Agency's *Energy Star Memorandum of Understanding #2* will have automatic `shutdown` enabled (as if *behavior* field were set to `shutdown),` and all others will not. See NOTES.

`unconfigured` The system will not be shut down automatically. If the system has just been installed or upgraded, the value of this field will be changed upon the next reboot.

You can use the following format to configure the system's notion of idleness:

idleness_parameter value

Where *idleness_parameter* can be:

`ttychars` If the *idleness_parameter* is `ttychars`, the *value* field will be interpreted as the maximum number of tty characters that can pass through the `ldterm` module while still allowing the system to be considered idle. This value defaults to `0` if no entry is provided.

`loadaverage` If the *idleness_parameter* is `loadaverage`, the (floating point) *value* field will be interpreted as the maximum load average that can be seen while still allowing the system to be considered idle. This value defaults to `0.04` if no entry is provided.

`diskreads` If the *idleness_parameter* is `diskreads`, the *value* field will be interpreted as the maximum number of disk reads that can be perform by the system while still allowing the system to be considered idle. This value defaults to `0` if no entry is provided.

`nfsreqs` — If the *idleness_parameter* is `nfsreqs`, the *value* field will be interpreted as the maximum number of NFS requests that can be sent or received by the system while still allowing the system to be considered idle. Null requests, access requests, and getattr requests are excluded from this count. This value defaults to `0` if no entry is provided.

`idlecheck` — If the *idleness_parameter* is `idlecheck`, the *value* must be pathname of a program to be executed to determine if the system is idle. If `autoshutdown` is enabled and the console keyboard, mouse, tty, CPU (as indicated by load average), network (as measured by NFS requests) and disk (as measured by read activity) have been idle for the amount of time specified in the `autoshutdown` entry specified above, and the time of day falls between the start and finish times, then this program will be executed to check for other idleness criteria. The *value* of the idle time specified in the above `autoshutdown` entry will be passed to the program in the environment variable `PM_IDLETIME.` The process must terminate with an exit code that represents the number of minutes that the process considers the system to have been idle.

There is no default *idlecheck* entry.

When the system is suspended, the current system state is saved on the disk in a statefile. An entry of following form can be used to change the location of statefile:

`statefile` *pathname*

where *pathname* identifies a block special file, for example, `/dev/dsk/c1t0d0s2`, or is the absolute pathname of a local ufs file. If the pathname specifies a block special file, it can be a symbolic link as long as it does not have a file system mounted on it. If pathname specifies a local ufs file, it cannot be a symbolic link. If the file does not exist, it will be created during the `suspend` operation. All the directory components of the path must already exist.

The actual size of statefile depends on a variety of factors, including the size of system memory, the number of loadable drivers / modules in use, the number and type of processes running, and the amount of user memory that has been locked down. It is recommended that statefile be placed on a file system with at least 10 Mbytes of free space. In case there is no statefile entry at boot time, an appropriate new entry is automatically created by the system.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWpmr
Interface stability	Evolving (Interfaces under DEVICE POWER MANAGEMENT are obsolete.)

SEE ALSO

`pmconfig`(1M), `powerd`(1M), `sys-unconfig`(1M), `uadmin`(2,) `attributes`(5), `cpr`(7), `ldterm`(7M), `pm`(7D)

Writing Device Drivers

Using Power Management

NOTES

SPARC desktop models first shipped after October 1, 1995 and before July 1, 1999 comply with the United States Enviromental Protection Agency's *Energy Star Memorandum of Understanding #2* guidelines and have `autoshutdown` enabled by default after 30 minutes of system idleness. This is achieved by `default` keyword of `autoshutdown` entry behave as `shutdown` for these machines. The user is prompted to confirm this default behavior at system installation reboot, or during the first reboot after the system is unconfigured by `sys-unconfig`(1M).

SPARC desktop models first shipped after July 1, 1999 comply with the United States Enviromental Protection Agency's *Energy Star Memorandum of Understanding #3* guidelines and have `autoshutdown` disabled by default, with `autopm` enabled after 30 minutes of idleness. This is achieved by interpreting default keyword of `autopm` entry behavior as `enabled` for these machines. User is not prompted to confirm this default behavior.

To determine the version of the EPA's *Energy Star Memorandum* applicable to your machine, use:

```
prtconf -pv | grep -i energystar
```

Absence of a property indicates no Energy Star guidelines are applicable to your machine.

System Power Management (suspend-resume) is currently supported only on a limited set of hardware platforms. Please see the book *Using Power Management* for a complete list of platforms that support system Power Management. See `uname`(2) to programatically determine if the machine supports suspend-resume.

NAME

printers – user-configurable printer alias database

SYNOPSIS

$HOME/.printers

DESCRIPTION

The `$HOME/.printers` file is a simplified version of the system `/etc/printers.conf` file (see `printers.conf`(4)). Users create the `$HOME/.printers` file in their home directory. This optional file is customizable by the user.

The `$HOME/.printers` file performs the following functions:

1. Sets personal aliases for all print commands.
2. Sets the interest list for the `lpget`, `lpstat`, and `cancel` commands. See `lpget`(1M), `lpstat`(1) and `cancel`(1).
3. Sets the default printer for the `lp`, `lpr`, `lpq`, and `lprm` commands. See `lp`(1), `lpr`(1B), `lpq`(1B), and `lprm`(1B).

Entries

Use a line or full screen editor to create or modify the `$HOME/.printers` file.

Each entry in `$HOME/.printers` describes one destination. Entries are one line consisting of two fields separated by either BLANKs or TABs and terminated by a NEWLINE. Format for an entry in `$HOME/.printers` varies according to the purpose of the entry.

Empty lines can be included for readability. Entries may continue on to multiple lines by adding a backslash ('\') as the last character in the line. The `$HOME/.printers` file can include comments. Comments have a pound sign ('#') as the first character in the line, and are terminated by a NEWLINE.

`Setting Personal Aliases`

Specify the alias or aliases in the first field. Separate multiple aliases by a pipe sign ('|'). Specify the destination in the second field. A destination names a printer or class of printers (see `lpadmin`(1M)). Specify the destination using atomic, POSIX-style (*server*:*destination*), or Federated Naming Service (FNS) (.../`service`/`printer`/...) names. See `printers.conf`(4) for information regarding the naming conventions for atomic and FNS names, and `standards`(5) for information regarding POSIX.

`Setting the Interest List for lpget, lpstat and cancel`

Specify `_all` in the first field. Specify the list of destinations for the interest list in the second field. Separate each destinations by a comma (','). Specify destinations using atomic, POSIX-style (*server*:*destination*), or FNS names (.../`service/printer/`...). See `printers.conf`(4) for information regarding the naming conventions for atomic and FNS names. This list of destinations may refer to an alias defined in `$HOME/.printers`.

Setting the Default Destination

Specify `_default` in the first field. Specify the default destination in the second field. Specify the default destination using atomic, POSIX-style (*server*`:`*destination*), or FNS names (...`/service/printer/`...). See `printers.conf`(4) for information regarding the naming conventions for atomic and FNS names. The default destination may refer to an alias defined in `$HOME/.printers`.

Locating Destination Information

The print client commands locate destination information based on the "printers" database entry in the `/etc/nsswitch.conf` file. See `nsswitch.conf`(4).

Locating the Personal Default Destination

The default destination is located differently depending on the command.

The `lp` command locates the default destination in the following order:

1. `lp` command's `–d` *destination* option.
2. `LPDEST` environment variable.
3. `PRINTER` environment variable.
4. `_default` destination in `$HOME/.printers`.
5. `_default` destination in `/etc/printers.conf`.
6. `_default` destination in FNS.

The `lpr`, `lpq`, and `lprm` commands locate the default destination in the following order:

1. `lpr` command's `–P` *destination* option.
2. `PRINTER` environment variable.
3. `LPDEST` environment variable.
4. `_default` destination in `$HOME/.printers`.
5. `_default` destination in `/etc/printers.conf`.
6. `_default` destination in FNS.

Locating the Interest List for lpget, lpstat, and cancel

The `lpget`, `lpstat`, and `cancel` commands locate the interest list in the following order:

1. `_all` list in `$HOME/.printers`.
2. `_all` list in `/etc/printers.conf`.
3. `_all` list in FNS.

EXAMPLES

EXAMPLE 1 Setting the interest list

The following entry sets the interest list to destinations `ps`, `secure`, and `dog` at server `west` and `finance_ps` at site `bldg2`:

```
_all  ps,secure,west:dog,site/bldg2/service/printer/finance_ps
```

EXAMPLE 2 Setting aliases to a printer

The following entry sets the aliases `ps`, `lp`, and `lw` to `sparc_printer`:

```
ps|lp|lw sparc_printer
```

EXAMPLE 3 Setting an alias as a default destination

The following entry sets the alias `pcl` to `hplj` and sets it as the default destination:

```
pcl|_default hplj
```

EXAMPLE 4 Setting an alias to a server destination

The following entry sets the alias `secure` to destination `catalpa` at server `tabloid`:

```
secure tabloid:catalpa
```

EXAMPLE 5 Setting an alias to a site destination

The following entry sets the alias `insecure` to destination `legal_ps` at site `bldg2`:

```
insecure site/bldg2/service/printer/legal_ps
```

FILES

`$HOME/.printers`	User-configurable printer database.
`/etc/printers.conf`	System printer configuration database.
`printers.conf.byname`	NIS version of `/etc/printers.conf`.
`printers.org_dir`	NIS+ version of `/etc/printers.conf`.
`fns.ctx_dir.`*domain*	FNS version of `/etc/printers.conf`.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWpcu
Stability Level	Stable

SEE ALSO

cancel(1), lp(1), lpq(1B), lpr(1B), lprm(1B), lpstat(1), lpadmin(1M), lpget(1M), nsswitch.conf(4), printers.conf(4), attributes(5), fns(5), standards(5)

System Administration Guide, Volume 1

NOTES

$HOME/.printers is referenced by the printing commands before further name resolution is made in /etc/printers.conf or the name service. If the alias references a destination defined in /etc/printers.conf, it is possible that the destination is defined differently on different systems. This could cause output to be sent to an unintended destination if the user is logged in to a different system.

NAME printers.conf – system printing configuration database

SYNOPSIS /etc/printers.conf

NIS printers.conf.byname

NIS+ printers.org_dir

FNS fns.ctx_dir.domain

DESCRIPTION The printers.conf file is the system printing configuration database. System administrators use printers.conf to describe destinations for the print client commands and the print protocol adaptor. A destination names a printer or class of printers (see lpadmin(1M)). The LP print spooler uses private LP configuration data for represented in the printers.conf database.

Entries Each entry in printers.conf describes one destination. Entries are one line consisting of any number of fields separated by colons (':') and terminated by a NEWLINE. The first field of each entry specifies the name of the destination and aliases to which the entry describes. Specify one or more names or aliases of the destination in this first field. Specify the destination using atomic names. POSIX-style names are not acceptable. See standards(5). Separate destination names by pipe signs ('|').

Two destination names are reserved for special use in the first entry. Use _all to specify the interest list for lpget, lpstat, and cancel. Use _default to specify the default destination.

The remaining fields in an entry are *key*=*value* pairs. See Specifying Configuration Options for details regarding *key*=*value* pairs.

Empty lines can be included for readability. Entries may continue on to multiple lines by adding a backslash ('\') as the last character in the line. printers.conf can include comments. Comments have a pound sign ('#') as the first character in the line, and are terminated by a NEWLINE. Use the lpset command to create or modify printers.conf (see lpset(1M)). *Do not make changes in* printers.conf *using an editor.*

Specifying Configuration Options *key*=*value* pairs are configuration options defined by the system administrator. *key* and *value* may be of arbitrary length. Separate *key* and *value* by the equal ('=') character.

Client/Server Configuration Options

The following client / server configuration options (represented as *key*=*value* pairs) are supported:

bsdaddr=*server*,*destination*[,Solaris]
Sets the server and destination name. Sets if the client generates protocol extensions for use with the lp command (see lp(1)). Solaris specifies a

Solaris print server extension. If `Solaris` is not specified, no protocol extensions are generated. *server* is the name of the host containing the queue for *destination*. *destination* is the atomic name by which the server knows the destination.

`use=`*destination*
Sets the destination to continue searching for configuration information. *destination* is an atomic or Federated Naming Service (FNS) (.../`service`/`printer`/...) name.

`all=`*destination_list*
Sets the interest list for the `lpget`, `lpstat`, and `cancel` commands. *destination_list* is a comma-separated list of destinations. Specify *destination* using atomic or FNS names (...`/service/printer/`...). See `lpget`(1M), `lpstat`(1), and `cancel`(1).

General Server Options

The following general server configuration options (represented as *key*=*value* pairs) are supported:

`spooling-type=`*spooler*[,*version*]
Sets the type of spooler under which a destination is configured. Dynamically loads translation support for the back-end spooling system from `/usr/lib/print/bsd-adaptor/bsd_spooler.so`[.*version*]. Specify *spooler* as `lpsched`, `cascade`, or `test`. `lpsched` is used as a default for locally attached destinations. `cascade` is used as a default for destination spooled on a remote host. Use `test` for the test module to allow the capture of print requests. If using a versioned spooler module, *version* specifies the version of the translation module.

`spooling-type-path=`*dir_list*
Sets the location of translation support for the type of spooler defined by the `spooling-type` key. Locates translation support for the for the type of spooler under which a destination is configured. *dir_list* is a comma-separated list of absolute pathnames to the directories used to locate translation support for the spooling system set by the `spooling-type` key.

LP Server Options

The following LP configuration options (represented as *key*=*value* pairs) are supported:

`user-equivalence=true|false`
Sets whether or not usernames are considered equivalent when cancelling a print request submitted from a different host in a networked environment. `true` means that usernames are considered equivalent, and permits users to cancel a print requests submitted from a different host. `user-equivalence` is set to `false` by default. `false` means that usernames are not considered

equivalent, and does not permit users cancel a print request submitted from a different host. If `user-equivalence` is set to `false`, print requests can only be cancelled by the users on the host on whichs the print prequest was generated or by the super-user on the print server.

Test Configuration Options

The following test configuration options (represented as *key*=*value* pairs) are supported:

`test-spooler-available=true|false`
: Sets whether or not the protocol adaptor accepts connection requests to the test adaptor for the destination. `true` means that the protocol adaptor accepts connection requests to the test adaptor for the destination. `test-spooler-available` is set to `true` by default. `false` means that the protocol adaptor does not accept connection requests to the test adaptor for the destination.

`test-log=`*dir*
: Sets the location of the log file generated by the test translation module. Specify *dir* as an absolute pathname.

`test-dir=`*dir*
: Sets the directory to be used during execution of the test translation module. Specify *dir* as an absolute pathname.

`test-access=true|false`
: Sets whether or not the requesting client has access to the test translation module. `true` means that the requesting client has access to the test translation module. `test-access` is set to `true` by default. `false` means that the the requesting client does not have access to the test translation module.

`test-accepting=true|false`
: Sets whether or not the configured destination is accepting job submission requests. `true` means that the configured destination is accepting job submission requests. `test-accepting` is set to `true` by default. `false` means that the configured destination is not accepting job submission requests.

`test-restart=true|false`
: Sets whether or not a protocol request to restart the destination will be honored or return an error. `true` means that a protocol request to restart the destination will be honored. `test-restart` is set to `true` by default. `false` means that a protocol request to restart the destination return an error.

`test-submit=true|false`

Sets whether or not a protocol request to submit a job to a destination will be honored or return an error. `true` means that a protocol request to submit a job to a destination will be honored. `test-submit` is set to `true` by default. `false` means that a protocol request to submit a job to a destination will not be honored.

`test-show-queue-file=`*file*
Sets the name of the file whose contents are to be returned as the result of a status query. Specify *file* as an absolute pathname.

`test-cancel-cancel-file=`*file*
Sets the name of the file whose contents are returned as the result of a cancellation request. Specify *file* as an absolute pathname.

Locating Destination Information

The print client commands and the print protocol adaptor locate destination information based on the “printers” database entry in the `/etc/nsswitch.conf` file. See `nsswitch.conf`(4).

`Locating the Personal Default Destination`

The default destination is located differently depending on the command.

The `lp` command locates the default destination in the following order:

1. `lp` command’s `–d` *destination* option.
2. `LPDEST` environment variable.
3. `PRINTER` environment variable.
4. `_default` destination in `$HOME/.printers`.
5. `_default` destination in `/etc/printers.conf`.
6. `_default` destination in FNS.

The `lpr`, `lpq`, and `lprm` commands locate the default destination in the following order:

1. `lpr` command’s `–P` *destination* option.
2. `PRINTER` environment variable.
3. `LPDEST` environment variable.
4. `_default` destination in `$HOME/.printers`.
5. `_default` destination in `/etc/printers.conf`.
6. `_default` destination in FNS.

`Locating the Interest List for lpstat, lpget, and cancel`

The `lpget`, `lpstat`, and `cancel` commands locate the interest list in the following order:

1. `_all` list in `$HOME/.printers`.
2. `_all` list in `/etc/printers.conf`.
3. `_all` list in FNS.

Looking Up Destinations Using Atomic Names and FNS

Federated Naming Service (FNS) supports resolution of *composite* names spanning multiple naming systems. FNS supports several underlying naming services: NIS+, NIS, and `files`.

Atomic destination names are resolved using the search order specified by the “printers” database entry in the `/etc/nsswitch.conf` file. When the “xfn” service is configured in the “printers” database, the following Federated Name Service contexts are searched for the supplied name:

```
thisuser/service/printer,
myorgunit/service/printer,
```

EXAMPLES

EXAMPLE 1 Setting the interest list

The following entry sets the interest list for the `lpget`, `lpstat` and `cancel` commands to `printer1`, `printer2` and `printer3`:

```
_all:all=printer1,printer2,printer3
```

EXAMPLE 2 Setting the server name

The following entry sets the server name to `server` and and printer name to `ps_printer` for destinations `printer1` and `ps`. It does not generate protocol extensions.

```
printer1|ps:bsdaddr=server,ps_printer
```

EXAMPLE 3 Setting server name and destination name

The following entry sets the server name to `server` and destination name to `pcl_printer`, for destination `printer2`. It also generates `Solaris` protocol extensions.

```
printer2:bsdaddr=server,pcl_printer,Solaris
```

EXAMPLE 4 Setting server name and destination name with continuous search

The following entry sets the server name to `server` and destination name to `new_printer`, for destination `printer3`. It also sets the `printer3` to continue searching for configuration information to printer `another_printer`.

```
printer3:bsdaddr=server,new_printer:use=another_printer
```

EXAMPLE 5 Setting default destination

The following entry sets the default destination to continue searching for configuration information to destination `printer1`.

```
_default:use=printer1
```

FILES

`/etc/printers.conf`
System configuration database.

`$HOME/.printers`
User-configurable printer database.

`printers.conf.byname` (NIS)
NIS version of `/etc/printers.conf`.

`printers.org_dir` (NIS+)
NIS+ version of `/etc/printers.conf`.

`fns.ctx_dir.`*domain*
FNS version of `/etc/printers.conf`.

`/usr/lib/print/bsd-adaptor/bsd_spooler.so*`
Spooler translation modules.

`/usr/lib/print/in.lpd`
BSD print protocol adapter.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWpcu
Stability Level	Stable

SEE ALSO

`cancel`(1), `lp`(1), `lpq`(1B), `lpr`(1B), `lprm`(1B), `lpstat`(1), `in.lpd`(1M), `lpadmin`(1M), `lpget`(1M), `lpset`(1M), `nsswitch.conf`(4), `printers`(4), `attributes`(5), `fns`(5), `fns_policies`(5), `standards`(5)

System Administration Guide, Volume 1

NAME

proc – /proc, the process file system

DESCRIPTION

/proc is a file system that provides access to the state of each process and light-weight process (lwp) in the system. The name of each entry in the /proc directory is a decimal number corresponding to a process-ID. These entries are themselves subdirectories. Access to process state is provided by additional files contained within each subdirectory; the hierarchy is described more completely below. In this document, "/proc file" refers to a non-directory file within the hierarchy rooted at /proc. The owner of each /proc file and subdirectory is determined by the user-ID of the process.

/proc can be mounted on any mount point, in addition to the standard /proc mount point, and can be mounted several places at once. Such additional mounts are allowed in order to facilitate the confinement of processes to subtrees of the file system via chroot(1M) and yet allow such processes access to commands like ps(1).

Standard system calls are used to access /proc files: open(2), close(2), read(2), and write(2) (including readv(2), writev(2), pread(2), and pwrite(2)). Most files describe process state and can only be opened for reading. ctl and lwpctl (control) files permit manipulation of process state and can only be opened for writing. as (address space) files contain the image of the running process and can be opened for both reading and writing. An open for writing allows process control; a read-only open allows inspection but not control. In this document, we refer to the process as open for reading or writing if any of its associated /proc files is open for reading or writing.

In general, more than one process can open the same /proc file at the same time. *Exclusive open* is an advisory mechanism provided to allow controlling processes to avoid collisions with each other. A process can obtain exclusive control of a target process, with respect to other cooperating processes, if it successfully opens any /proc file in the target process for writing (the as or ctl files, or the lwpctl file of any lwp) while specifying O_EXCL in the open(2). Such an open will fail if the target process is already open for writing (that is, if an as, ctl, or lwpctl file is already open for writing). There can be any number of concurrent read-only opens; O_EXCL is ignored on opens for reading. It is recommended that the first open for writing by a controlling process use the O_EXCL flag; multiple controlling processes usually result in chaos.

If a process opens one of its own /proc files for writing, the open succeeds regardless of O_EXCL and regardless of whether some other process has the process open for writing. Self-opens do not count when another process attempts an exclusive open. (A process cannot exclude a debugger by opening itself for writing and the application of a debugger cannot prevent a process from opening itself.) All self-opens for writing are forced to be close-on-exec (see the F_SETFD operation of fcntl(2)).

Data may be transferred from or to any locations in the address space of the traced process by applying `lseek`(2) to position the `as` file at the virtual address of interest followed by `read`(2) or `write`(2) (or by using `pread`(2) or `pwrite`(2) for the combined operation). The address-map file `/proc/`*pid*`/map` can be read 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. A write request beginning at an unmapped virtual address fails with `EIO`; a read request beginning at an unmapped virtual address returns zero (an end-of-file indication).

Information and control operations are provided through additional files. `<procfs.h>` contains definitions of data structures and message formats used with these files. Some of these definitions 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`.

Every 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. All lwps in a process share its address space as well as many other attributes. Through the use of `lwpctl` and `ctl` files as described below, it is possible to affect individual lwps in a process or to affect all of them at once, depending on the operation.

When the process has more than one lwp, a representative lwp is chosen by the system for certain process status files and control operations. The representative 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 lwps 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). See the description of the `status` file for definitions of stopped states. See the PCSTOP control operation for the definition of "event of interest".

The representative lwp remains fixed (it will be chosen again on the next operation) as long as all of the lwps are stopped on events of interest or are in a PR_SUSPENDED stop and the PCRUN control operation is not applied to any of them.

When applied to the process control file, every /proc control operation that must act on an lwp uses the same algorithm to choose which lwp to act upon. Together with synchronous stopping (see PCSET), this enables a debugger to control a multiple-lwp process using only the process-level status and control files if it so chooses. More fine-grained control can be achieved using the lwp-specific files.

The system supports two process data models, the traditional 32-bit data model in which ints, longs and pointers are all 32 bits wide (the ILP32 data model), and on some platforms the 64-bit data model in which longs and pointers, but not ints, are 64 bits in width (the LP64 data model). In the LP64 data model some system data types, notably size_t, off_t, time_t and dev_t, grow from 32 bits to 64 bits as well.

The /proc interfaces described here are available to both 32-bit and 64-bit controlling processes. However, many operations attempted by a 32-bit controlling process on a 64-bit target process will fail with EOVERFLOW because the address space range of a 32-bit process cannot encompass a 64-bit process or because the data in some 64-bit system data type cannot be compressed to fit into the corresponding 32-bit type without loss of information. Operations that fail in this circumstance include reading and writing the address space, reading the address-map file, and setting the target process's registers. There is no restriction on operations applied by a 64-bit process to either a 32-bit or a 64-bit target processes.

The format of the contents of any /proc file depends on the data model of the observer (the controlling process), not on the data model of the target process. A 64-bit debugger does not have to translate the information it reads from a /proc file for a 32-bit process from 32-bit format to 64-bit format. However, it usually has to be aware of the data model of the target process. The pr_dmodel field of the status files indicates the target process's data model.

To help deal with system data structures that are read from 32-bit processes, a 64-bit controlling program can be compiled with the C preprocessor symbol _SYSCALL32 defined before system header files are included. This makes explicit 32-bit fixed-width data structures (like cstruct stat32) visible to the 64-bit program. See types32(3HEAD).

DIRECTORY STRUCTURE

At the top level, the directory /proc contains entries each of which names an existing process in the system. These entries are themselves directories. Except where otherwise noted, the files described below can be opened for reading only. In addition, if a process becomes a *zombie* (one that has exited but whose parent has not yet performed a wait(2) upon it), most of its associated /proc files disappear from the hierarchy; subsequent attempts to open them, or to read or write files opened before the process exited, will elicit the error ENOENT.

Although process state and consequently the contents of /proc files can change from instant to instant, a single read(2) of a /proc file is guaranteed to return a sane representation of state; that is, the read will be atomic with respect to the state of the process. No such guarantee applies to successive reads applied to a /proc file for a running process. In addition, atomicity is not guaranteed for I/O applied to the as (address-space) file for a running process or for a process whose address space contains memory shared by another running process.

A number of structure definitions are used to describe the files. These structures may grow by the addition of elements at the end in future releases of the system and it is not legitimate for a program to assume that they will not.

STRUCTURE OF /proc/*pid*

A given directory /proc/*pid* contains the following entries. A process can use the invisible alias /proc/self if it wishes to open one of its own /proc files (invisible in the sense that the name “self” does not appear in a directory listing of /proc obtained from ls(1), getdents(2), or readdir(3C)).

as
Contains the address-space image of the process; it can be opened for both reading and writing. lseek(2) is used to position the file at the virtual address of interest and then the address space can be examined or changed through read(2) or write(2) (or by using pread(2) or pwrite(2) for the combined operation).

ctl
A write-only file to which structured messages are written directing the system to change some aspect of the process’s state or control its behavior in some way. The seek offset is not relevant when writing to this file. Individual lwps also have associated lwpctl files in the lwp subdirectories. A control message may be written either to the process’s ctl file or to a specific lwpctl file with operation-specific effects. The effect of a control message is immediately reflected in the state of the process visible through appropriate status and information files. The types of control messages are described in detail later. See CONTROL MESSAGES.

status
Contains state information about the process and the representative lwp. The file contains a pstatus structure which contains an embedded lwpstatus structure for the representative lwp, as follows:

```
typedef struct pstatus {
   int pr_flags;            /* flags (see below) */
   int pr_nlwp;             /* number of lwps in the process */
   pid_tpr_pid;             /* process id */
   pid_tpr_ppid;            /* parent process id */
   pid_tpr_pgid;            /* process group id */
   pid_tpr_sid;             /* session id */
   id_t pr_aslwpid;         /* lwp-id of the aslwp, if any */
   id_t pr_agentid;         /* lwp-id of the agent lwp, if any */
   sigset_t pr_sigpend;     /* set of process pending signals */
   uintptr_t pr_brkbase;    /* virtual address of the process heap */
```

```
    size_t pr_brksize;          /* size of the process heap, in bytes */
    uintptr_t pr_stkbase;       /* virtual address of the process stack */
    size_tpr_stksize;           /* size of the process stack, in bytes */
    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 r_cstime;       /* sum of children’s system times */
    sigset_t pr_sigtrace;       /* set of traced signals */
    fltset_t pr_flttrace;       /* set of traced faults */
    sysset_t pr_sysentry;       /* set of system calls traced on entry */
    sysset_t pr_sysexit;        /* set of system calls traced on exit */
    char pr_dmodel;             /* data model of the process */
    lwpstatus_t pr_lwp;         /* status of the representative lwp */
} pstatus_t;
```

pr_flags is a bit-mask holding the following process flags. For convenience, it also contains the lwp flags for the representative lwp, described later.

PR_ISSYS process is a system process (see PCSTOP).

PR_VFORKP process is the parent of a vforked child (see PCWATCH).

PR_FORK process has its inherit-on-fork mode set (see PCSET).

PR_RLC process has its run-on-last-close mode set (see PCSET).

PR_KLC process has its kill-on-last-close mode set (see PCSET).

PR_ASYNC process has its asynchronous-stop mode set (see PCSET).

PR_MSACCT process has microstate accounting enabled (see PCSET).

PR_MSFORK process microstate accounting is inherited on fork (see PCSET).

PR_BPTADJ process has its breakpoint adjustment mode set (see PCSET).

PR_PTRACE process has its ptrace-compatibility mode set (see PCSET).

pr_nlwp is the total number of lwps in the process.

pr_pid, pr_ppid, pr_pgid, 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_aslwpid is the lwp-ID for the "asynchronous signal lwp" (aslwp). It is zero if there is no aslwp in the process. The aslwp is the lwp designated to redirect asynchronous signals to other lwps in a multi-threaded process. See signal(3HEAD) for a description of the aslwp.

pr_agentid is the lwp-ID for the /proc agent lwp (see the PCAGENT control operation). It is zero if there is no agent lwp in the process.

pr_sigpend identifies asynchronous signals pending for the process.

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_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_sigtrace and pr_flttrace contain, respectively, the set of signals and the set of hardware faults that are being traced (see PCSTRACE and PCSFAULT).

pr_sysentry and pr_sysexit contain, respectively, the sets of system calls being traced on entry and exit (see PCSENTRY and PCSEXIT).

pr_dmodel indicates the data model of the process. Possible values are:

PR_MODEL_ILP32	process data model is ILP32.
PR_MODEL_LP64	process data model is LP64.
PR_MODEL_NATIVE	process data model is native.

The constant PR_MODEL_NATIVE reflects the data model of the controlling process, *that is*, its value is PR_MODEL_ILP32 or PR_MODEL_LP64 according to whether the controlling process has been compiled as a 32-bit program or a 64-bit program, respectively.

pr_lwp contains the status information for the representative lwp:

```
typedef struct lwpstatus {
 int pr_flags;               /* flags (see below) */
 id_t pr_lwpid;              /* specific lwp identifier */
 short pr_why;               /* reason for lwp stop, if stopped */
 short pr_what;              /* more detailed reason */
 short pr_cursig;            /* current signal, if any */
 siginfo_t pr_info;          /* info associated with signal or fault */
 sigset_t pr_lwppend;        /* set of signals pending to the lwp */
 sigset_t pr_lwphold;        /* set of signals blocked by the lwp */
 struct sigaction pr_action; /* signal action for current signal */
 stack_t pr_altstack;        /* alternate signal stack info */
 uintptr_t pr_oldcontext;    /* address of previous ucontext */
 short pr_syscall;           /* system call number (if in syscall) */
 short pr_nsysarg;           /* number of arguments to this syscall */
 int pr_errno;               /* errno for failed syscall */
 long pr_sysarg[PRSYSARGS];  /* arguments to this syscall */
 long pr_rval1;              /* primary syscall return value */
 long pr_rval2;              /* second syscall return value, if any */
 char pr_clname[PRCLSZ];     /* scheduling class name */
 timestruc_t pr_tstamp;      /* real-time time stamp of stop */
 ulong_t pr_instr;           /* current instruction */
```

```
 prgregset_t pr_reg;          /* general registers */
 prfpregset_t pr_fpreg;       /* floating-point registers */
} lwpstatus_t;
```

`pr_flags` is a bit-mask holding the following lwp flags. For convenience, it also contains the process flags, described previously.

PR_STOPPED lwp is stopped.

PR_ISTOP lwp is stopped on an event of interest (see PCSTOP).

PR_DSTOP lwp has a stop directive in effect (see PCSTOP).

PR_STEP lwp has a single-step directive in effect (see PCRUN).

PR_ASLEEP lwp is in an interruptible sleep within a system call.

PR_PCINVAL lwp's current instruction (`pr_instr`) is undefined.

PR_ASLWP this is the asynchronous signal lwp for the process.

PR_AGENT this is the `/proc` agent lwp for the process.

`pr_lwpid` names the specific lwp.

`pr_why` and `pr_what` together describe, for a stopped lwp, the reason for the stop. Possible values of `pr_why` and the associated `pr_what` are:

PR_REQUESTED indicates that the stop occurred in response to a stop directive, normally because PCSTOP was applied or because another lwp stopped on an event of interest and the asynchronous-stop flag (see PCSET) 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 PCSTRACE); `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 PCSFAULT); `pr_what` holds the fault number that caused the stop.

PR_SYSENTRY
PR_SYSEXIT indicate a stop on entry to or exit from a system call (see PCSENTRY and PCSEXIT); `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 lwps within the process. pr_what is unused in this case.

pr_cursig names the current signal, that is, the next signal to be delivered to the lwp, if any. 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_lwppend identifies any synchronous or directed signals pending for the lwp. pr_lwphold identifies those signals whose delivery is being blocked by the lwp (the signal mask).

pr_action contains the signal action information pertaining to the current signal (see sigaction(2)); it is undefined if pr_cursig is zero. pr_altstack contains the alternate signal stack information for the lwp (see sigaltstack(2)).

pr_oldcontext, if not zero, contains the address on the lwp stack of a ucontext structure describing the previous user-level context (see ucontext(3HEAD)). It is non-zero only if the lwp is executing in the context of a signal handler.

pr_syscall is the number of the system call, if any, being executed by the lwp; it is non-zero if and 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 pr_sysarg contains the actual arguments.

pr_rval1, pr_rval2, and pr_errno are defined only if the lwp is stopped on PR_SYSEXIT or if the PR_VFORKP flag is set. If pr_errno is zero, pr_rval1 and pr_rval2 contain the return values from the system call. Otherwise, pr_errno contains the error number for the failing system call (see <sys/errno.h>).

pr_clname contains the name of the lwp's scheduling class.

pr_tstamp, if the lwp is stopped, contains a time stamp marking when the lwp stopped, in real time seconds and nanoseconds since an arbitrary time in the past.

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 based machines, it is a 32-bit word. On IA based machines, it is a single byte. 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.

pr_reg is an array holding the contents of a stopped lwp's general registers.

SPARC On SPARC-based machines, the predefined constants R_G0 ... R_G7, R_O0 ... R_O7, R_L0 ...

R_L7, R_I0 ... R_I7, R_PC, R_nPC, and R_Y can be used as indices to refer to the corresponding registers; previous register windows can be read from their overflow locations on the stack (however, see the gwindows file in the /proc/*pid*/lwp/*lwpid* subdirectory).

SPARC V8 (32-bit) — For SPARC V8 (32-bit) controlling processes, the predefined constants R_PSR, R_WIM, and R_TBR can be used as indices to refer to the corresponding special registers. For SPARC V9 (64-bit) controlling processes, the predefined constants R_CCR, R_ASI, and R_FPRS can be used as indices to refer to the corresponding special registers.

IA — On IA based 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.

pr_fpreg is a structure holding the contents of the floating-point registers.

SPARC registers, both general and floating-point, as seen by a 64-bit controlling process are the V9 versions of the registers, even if the target process is a 32-bit (V8) process. V8 registers are a subset of the V9 registers.

If the lwp is not stopped, all register values are undefined.

psinfo — Contains miscellaneous information about the process and the representative lwp needed by the ps(1) command. psinfo is accessible after a process becomes a *zombie*. The file contains a psinfo structure which contains an embedded lwpsinfo structure for the representative lwp, as follows:

```
typedef struct psinfo {
 int pr_flag;              /* process flags */
 int pr_nlwp;              /* number of lwps in the process */
 pid_t pr_pid;             /* process id */
 pid_t pr_ppid;            /* process id of parent */
 pid_t pr_pgid;            /* process id of process group leader */
 pid_t pr_sid;             /* session id */
 uid_t pr_uid;             /* real user id */
 uid_t pr_euid;            /* effective user id */
 gid_t pr_gid;             /* real group id */
 gid_t pr_egid;            /* effective group id */
 uintptr_t pr_addr;        /* address of process */
 size_t pr_size;           /* size of process image in Kbytes */
 size_t pr_rssize;         /* resident set size in Kbytes */
 dev_t pr_ttydev;          /* controlling tty device (or PRNODEV) */
```

```
 ushort_t pr_pctcpu;       /* % of recent cpu time used by all lwps */
 ushort_t pr_pctmem;       /* % of system memory used by process */
 timestruc_t pr_start;     /* process start time, from the epoch */
 timestruc_t pr_time;      /* cpu time for this process */
 timestruc_t pr_ctime;     /* cpu time for reaped children */
 char pr_fname[PRFNSZ];    /* name of exec'ed file */
 char pr_psargs[PRARGSZ];  /* initial characters of arg list */
 int pr_wstat;             /* if zombie, the wait() status */
 int pr_argc;              /* initial argument count */
 uintptr_t pr_argv;        /* address of initial argument vector */
 uintptr_t pr_envp;        /* address of initial environment vector */
 char pr_dmodel;           /* data model of the process */
 lwpsinfo_t pr_lwp;        /* information for representative lwp */
} psinfo_t;
```

Some of the entries in `psinfo`, such as `pr_flag` and `pr_addr`, refer to internal kernel data structures and should not be expected to retain their meanings across different versions of the operating system.

`pr_pctcpu` and `pr_pctmem` are 16-bit binary fractions in the range 0.0 to 1.0 with the binary point to the right of the high-order bit (1.0 == 0x8000). `pr_pctcpu` is the summation over all lwps in the process.

`pr_lwp` contains the `ps`(1) information for the representative lwp. If the process is a *zombie*, `pr_nlwp` and `pr_lwp.pr_lwpid` are zero and the other fields of `pr_lwp` are undefined:

```
typedef struct lwpsinfo {
 int pr_flag;              /* lwp flags */
 id_t pr_lwpid;            /* lwp id */
 uintptr_t pr_addr;        /* internal address of lwp */
 uintptr_t pr_wchan;       /* wait addr for sleeping lwp */
 char pr_stype;            /* synchronization event type */
 char pr_state;            /* numeric lwp state */
 char pr_sname;            /* printable character for pr_state */
 char pr_nice;             /* nice for cpu usage */
 short pr_syscall;         /* system call number (if in syscall) */
 char pr_oldpri;           /* pre-SVR4, low value is high priority */
 char pr_cpu;              /* pre-SVR4, cpu usage for scheduling */
 int pr_pri;               /* priority, high value = high priority */
 ushort_t pr_pctcpu;       /* % of recent cpu time used by this lwp */
 timestruc_t pr_start;     /* lwp start time, from the epoch */
 timestruc_t pr_time;      /* cpu time for this lwp */
 char pr_clname[PRCLSZ];   /* scheduling class name */
 char pr_name[PRFNSZ];     /* name of system lwp */
 processorid_t pr_onpro;   /* processor which last ran this lwp */
 processorid_t pr_bindpro; /* processor to which lwp is bound */
 psetid_t pr_bindpset;     /* processor set to which lwp is bound */
} lwpsinfo_t;
```

Some of the entries in `lwpsinfo`, such as `pr_flag`, `pr_addr`, `pr_wchan`, `pr_stype`, `pr_state`, and `pr_name`, refer to internal kernel data structures and should not be expected to retain their meanings across different versions of the operating system.

`pr_pctcpu` is a 16-bit binary fraction, as described above. It represents the CPU time used by the specific lwp. On a multi-processor machine, the maximum value is 1 / N, where N is the number of CPUs.

cred Contains a description of the credentials associated with the process:

```
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) */
 int pr_ngroups;     /* number of supplementary groups */
 gid_t pr_groups[1]; /* array of supplementary groups */
} prcred_t;
```

The array of associated supplementary groups in `pr_groups` is of variable length; the `cred` file contains all of the supplementary groups. `pr_ngroups` indicates the number of supplementary groups. (See also the `PCSCRED` control operation.)

sigact Contains an array of `sigaction structures` describing the current dispositions of all signals associated with the traced process (see `sigaction`(2)). 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.

auxv Contains the initial values of the process's aux vector in an array of `auxv_t` structures (see `<sys/auxv.h>`). The values are those that were passed by the operating system as startup information to the dynamic linker.

ldt This file exists only on IA based machines. It is non-empty only if the process has established a local descriptor table (LDT). If non-empty, the file contains the array of currently active LDT entries in an array of elements of type `struct ssd`, defined in `<sys/sysi86.h>`, one element for each active LDT entry.

map Contains information about the virtual address map of the process. The file contains an array of `prmap` structures, each of which describes a contiguous virtual address region in the address space of the traced process:

```
typedef struct prmap {
 uintptr_tpr_vaddr;         /* virtual address of mapping */
 size_t pr_size;            /* size of mapping in bytes */
 char pr_mapname[PRMAPSZ];  /* name in /proc/pid/object */
 offset_t pr_offset;        /* offset into mapped object, if any */
 int pr_mflags;             /* protection and attribute flags */
 int pr_pagesize;           /* pagesize for this mapping in bytes */
```

```
 int pr_shmid;              /* SysV shared memory identifier */
} prmap_t;
```

`pr_vaddr` is the virtual address of the mapping within the traced process and `pr_size` is its size in bytes. `pr_mapname`, if it does not contain a null string, contains the name of a file in the `object` directory (see below) that can be opened read-only to obtain a file descriptor for the mapped file associated with the mapping. This enables a debugger to find object file symbol tables without having to know the real path names of the executable file and shared libraries of the process. `pr_offset` is the 64-bit offset within the mapped file (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_ISM`	mapping is intimate shared memory (shared MMU resources).

A contiguous area of the address space having the same underlying mapped object may appear as multiple mappings due to varying read, write, and execute 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 fail.

`pr_pagesize` is the page size for the mapping, currently always the system pagesize.

`pr_shmid` is the shared memory identifier, if any, for the mapping. Its value is –`1` if the mapping is not System V shared memory. See `shmget`(2).

rmap Contains information about the reserved address ranges of the process. The file contains an array of `prmap` structures, as defined above for the `map` file. Each structure describes a contiguous virtual address region in the address space of the traced process that is reserved by the system in the sense that an `mmap`(2) system call that does not specify `MAP_FIXED` will not use any part of it for the new mapping. Examples of such reservations include the address ranges reserved for the process stack and the individual thread stacks of a multi-threaded process.

cwd A symbolic link to the process's current working directory (see chdir(2)). A readlink(2) of /proc/*pid*/cwd yields a null string. However, it can be opened, listed, and searched as a directory and can be the target of chdir(2).

root A symbolic link to the process's root directory. /proc/*pid*/root can differ from the system root directory if the process or one of its ancestors executed chroot(2) as super-user. It has the same semantics as /proc/*pid*/cwd.

fd A directory containing references to the open files of the process. Each entry is a decimal number corresponding to an open file descriptor in the process.

If an entry refers to a regular file, it can be opened with normal file system semantics but, to ensure that the controlling process cannot gain greater access than the controlled process, with no file access modes other than its read / write open modes in the controlled process. If an entry refers to a directory, it appears as a symbolic link and can be accessed with the same semantics as /proc/*pid*/cwd. An attempt to open any other type of entry fails with EACCES.

object A directory containing read-only files with names corresponding to the pr_mapname entries in the map and pagedata files. Opening such a file yields a file descriptor for the underlying mapped file associated with an address-space mapping in the process. The file name a.out appears in the directory as an alias for the process's executable file.

The object directory makes it possible for a controlling process to gain access to the object file and any shared libraries (and consequently the symbol tables) without having to know the actual path names of the executable files.

pagedata Opening the page data file enables 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, time of read() */
 ulong_t pr_nmap;       /* number of address space mappings */
 ulong_t 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 {
 uintptr_t pr_vaddr;        /* virtual address of mapping */
 ulong_t pr_npage;          /* number of pages in mapping */
 char pr_mapname[PRMAPSZ];  /* name in /proc/pid/object */
 offset_t pr_offset;        /* offset into mapped object, if any */
 int pr_mflags;             /* protection and attribute flags */
 int pr_pagesize;           /* pagesize for this mapping in bytes */
 int pr_shmid;              /* SysV shared memory identifier */
} prasmap_t;
```

Each section header is followed by `pr_npage` bytes, one byte for each page in the mapping, plus 0-7 null bytes at the end so that the next `prasmap` structure begins on an eight-byte 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; every read starts from the beginning of the file. 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. An open of the page data file will fail with `ENOMEM` if the system-imposed limit would be exceeded.

watch Contains an array of `prwatch` structures, one for each watched area established by the `PCWATCH` control operation. See `PCWATCH` for details.

usage Contains process usage information described by a `prusage` structure which contains at least the following fields:

```
typedef struct prusage {
 id_tpr_lwpid;            /* lwp id.  0: process or defunct */
 int pr_count;            /* number of contributing lwps */
 timestruc_t pr_tstamp;   /* real time stamp, time of read() */
 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 */
 ulong_t pr_minf;         /* minor page faults */
 ulong_t pr_majf;         /* major page faults */
 ulong_t pr_nswap;        /* swaps */
 ulong_t pr_inblk;        /* input blocks */
 ulong_t pr_oublk;        /* output blocks */
 ulong_t pr_msnd;         /* messages sent */
 ulong_t pr_mrcv;         /* messages received */
 ulong_t pr_sigs;         /* signals received */
 ulong_t pr_vctx;         /* voluntary context switches */
 ulong_t pr_ictx;         /* involuntary context switches */
 ulong_t pr_sysc;         /* system calls */
 ulong_t pr_ioch;         /* chars read and written */
} prusage_t;
```

If microstate accounting has not been enabled for the process (see the `PR_MSACCT` flag for the `PCSET` operation, below), the `usage` file contains only an estimate of times spent in the various states. The `usage` file is accessible after a process becomes a *zombie*.

lstatus

Contains a `prheader` structure followed by an array of `lwpstatus` structures, one for each lwp in the process (see also `/proc/`*pid*`/lwp/`*lwpid*`/lwpstatus`, below). The `prheader` structure describes the number and size of the array entries that follow.

```
typedef struct prheader {
 long pr_nent;        /* number of entries */
 size_t pr_entsize;   /* size of each entry, in bytes */
} prheader_t;
```

The `lwpstatus` structure may grow by the addition of elements at the end in future releases of the system. Programs must use `pr_entsize` in the file header to index through the array. These comments apply to all `/proc` files that include a `prheader` structure (`lpsinfo` and `lusage`, below).

lpsinfo

Contains a `prheader` structure followed by an array of `lwpsinfo` structures, one for each lwp in the process. (See also `/proc/`*pid*`/lwp/`*lwpid*`/lwpsinfo`, below.)

lusage

Contains a `prheader` structure followed by an array of `prusage` structures, one for each lwp in the process plus an additional element at the beginning that contains the summation over all defunct lwps (lwps 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 obtained from the `usage` file. (See also `/proc/`*pid*`/lwp/`*lwpid*`/lwpusage`, below.)

lwp
A directory containing entries each of which names an lwp within the process. These entries are themselves directories containing additional files as described below.

STRUCTURE OF /proc/*pid*/lwp/*lwpid*

A given directory /proc/*pid*/lwp/*lwpid* contains the following entries:

lwpctl
Write-only control file. The messages written to this file affect the specific lwp rather than the representative lwp, as is the case for the process's ctl file.

lwpstatus
lwp-specific state information. This file contains the lwpstatus structure for the specific lwp as described above for the representative lwp in the process's status file.

lwpsinfo
lwp-specific ps(1) information. This file contains the lwpsinfo structure for the specific lwp as described above for the representative lwp in the process's psinfo file.

lwpusage
This file contains the prusage structure for the specific lwp as described above for the process's usage file.

gwindows
This file exists only on SPARC based machines. If it is non-empty, it contains a gwindows_t structure, defined in <sys/regset.h>, with the values 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 lwp is not stopped or if there are no register windows that could not be stored on the stack, the file is empty (the usual case).

xregs
Extra state registers. The extra state register set is architecture dependent; this file is empty if the system does not support extra state registers. If the file is non-empty, it contains an architecture dependent structure of type prxregset_t, defined in <procfs.h>, with the values of the lwp's extra state registers. If the lwp is not stopped, all register values are undefined. See also the PCSXREG control operation, below.

asrs
This file exists only for 64-bit SPARC V9 processes. It contains an asrset_t structure, defined in <sys/regset.h>, containing the values of the lwp's platform-dependent ancillary state registers. If the lwp is not stopped, all register values are undefined. See also the PCSASRS control operation, below.

CONTROL MESSAGES

Process state changes are effected through messages written to a process's ctl file or to an individual lwp's lwpctl file. All control messages consist of a long that names the specific operation followed by additional data containing the operand, if any.

Multiple control messages may be combined in a single `write`(2) (or `writev`(2)) to a control file, but no partial writes are permitted. That is, each control message, operation code plus operand, if any, must be presented in its entirety to the `write`(2) and not in pieces over several system calls. If a control operation fails, no subsequent operations contained in the same `write`(2) are attempted.

Descriptions of the allowable control messages follow. In all cases, writing a message to a control file for a process or lwp that has terminated elicits the error `ENOENT`.

PCSTOP PCDSTOP PCWSTOP PCTWSTOP

When applied to the process control file, `PCSTOP` directs all lwps to stop and waits for them to stop, `PCDSTOP` directs all lwps to stop without waiting for them to stop, and `PCWSTOP` simply waits for all lwps to stop. When applied to an lwp control file, `PCSTOP` directs the specific lwp to stop and waits until it has stopped, `PCDSTOP` directs the specific lwp to stop without waiting for it to stop, and `PCWSTOP` simply waits for the specific lwp to stop. When applied to an lwp control file, `PCSTOP` and `PCWSTOP` complete when the lwp stops on an event of interest, immediately if already so stopped; when applied to the process control file, they complete when every lwp has stopped either on an event of interest or on a `PR_SUSPENDED` stop.

`PCTWSTOP` is identical to `PCWSTOP` except that it enables the operation to time out, to avoid waiting forever for a process or lwp that may never stop on an event of interest. `PCTWSTOP` takes a `long` operand specifying a number of milliseconds; the wait will terminate successfully after the specified number of milliseconds even if the process or lwp has not stopped; a timeout value of zero makes the operation identical to `PCWSTOP`.

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 `PCSTRACE`, `PCSFAULT`, `PCSENTRY`, and `PCSEXIT`). `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 `PCSTOP` or `PCDSTOP` 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.

A write of a control message that blocks is interruptible by a signal 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 `PCSTOP` is interrupted, the lwp stop directives remain in effect even though the `write`(2) returns an error. (Use of `PCTWSTOP` with a non-zero timeout is recommended over `PCWSTOP` with an `alarm`(2).)

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 one of these operations to a system process or any of its lwps elicits the error EBUSY.

PCRUN Make an lwp runnable again after a stop. This operation takes a `long` operand containing zero or more of the following flags:

PRCSIG clears the current signal, if any (see PCCSIG).

PRCFAULT clears the current fault, if any (see PCCFAULT).

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 is not blocked, 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 IA based 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 PCSENTRY and PCSEXIT).

PRSTOP directs the lwp to stop again as soon as possible after resuming execution (see PCDSTOP). 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 control file, PCRUN clears any outstanding directed-stop request and makes the specific lwp runnable. The operation fails with EBUSY if the specific lwp is not stopped on an event of interest or has not been directed to stop or if the agent lwp exists and this is not the agent lwp (see PCAGENT).

When applied to the process control file, a representative lwp is chosen for the operation as described for /proc/*pid*/status. The operation fails with EBUSY if the representative lwp is not stopped on an event of interest or has not been directed to stop or if the agent lwp exists. If PRSTEP or PRSTOP was requested, the representative lwp is made runnable and its outstanding directed-stop request is cleared; otherwise all outstanding directed-stop requests are cleared and, if it was stopped on an event of interest, the representative lwp is marked PR_REQUESTED. If, as a consequence, all lwps are in the PR_REQUESTED or PR_SUSPENDED stop state, all lwps showing PR_REQUESTED are made runnable.

PCSTRACE Define 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 using an operand `sigset_t` contained in the control message. 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 (the signal mask) 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 `PCSHOLD`.

PCCSIG The current signal, if any, is cleared from the specific or representative lwp.

PCSSIG The current signal and its associated signal information for the specific or representative lwp are set according to the contents of the operand `siginfo` structure (see `<sys/siginfo.h>`). If the specified signal number is zero, the current signal is cleared. The semantics of this operation are different from those of `kill`(2) in that the signal is delivered to the lwp immediately after execution is resumed (even if it is being blocked) and an additional `PR_SIGNALLED` stop does not intervene even if the signal is traced. Setting the current signal to `SIGKILL` terminates the process immediately.

PCKILL If applied to the process control file, a signal is sent to the process with semantics identical to those of `kill`(2). If applied to an lwp control file, a directed signal is sent to the specific lwp. The signal is named in a `long` operand contained in the message. Sending `SIGKILL` terminates the process immediately.

PCUNKILL A signal is deleted, that is, it is removed from the set of pending signals. If applied to the process control file, the signal is deleted from the process's pending signals. If applied to an lwp control file, the signal is deleted from the lwp's pending signals. The current signal (if any) is unaffected. The signal is named in a `long` operand in the control message. It is an error (`EINVAL`) to attempt to delete `SIGKILL`.

PCSHOLD Set the set of held signals for the specific or representative lwp (signals whose delivery will be blocked if sent to the lwp). The set of signals is specified with a `sigset_t` operand. `SIGKILL` and `SIGSTOP` cannot be held; if specified, they are silently ignored.

PCSFAULT Define 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 the operand `fltset_t` structure. 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 (single-step)
FLTWATCH	watchpoint 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 PCCFAULT or by the PRCFAULT option of PCRUN. FLTPAGE is an exception; no signal is posted. The pr_info field in the lwpstatus structure identifies the signal to be sent and contains machine-specific information about the fault.

PCCFAULT

The current fault, if any, is cleared; the associated signal will not be sent to the specific or representative lwp.

PCSENTRY PCSEXIT

These control operations instruct the process's lwps to stop on entry to or exit from specified system calls. The set of system calls to be traced is defined via an operand sysset_t structure.

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 PCRUN control message. Unless exit from the system call is being traced, the lwp returns to user level showing EINTR.

PCWATCH

Set or clear a watched area in the controlled process from a prwatch structure operand:

```
typedef struct prwatch {
 uintptr_t pr_vaddr;  /* virtual address of watched area */
 size_t pr_size;      /* size of watched area in bytes */
```

```
 int pr_wflags;       /* watch type flags */
} prwatch_t;
```

pr_vaddr specifies the virtual address of an area of memory to be watched in the controlled process. pr_size specifies the size of the area, in bytes. pr_wflags specifies the type of memory access to be monitored as a bit-mask of the following flags:

WA_READ	read access
WA_WRITE	write access
WA_EXEC	execution access
WA_TRAPAFTER	trap after the instruction completes

If pr_wflags is non-empty, a watched area is established for the virtual address range specified by pr_vaddr and pr_size. If pr_wflags is empty, any previously-established watched area starting at the specified virtual address is cleared; pr_size is ignored.

A watchpoint is triggered when an lwp in the traced process makes a memory reference that covers at least one byte of a watched area and the memory reference is as specified in pr_wflags. When an lwp triggers a watchpoint, it incurs a watchpoint trap. If FLTWATCH is being traced, the lwp stops; otherwise, it is sent a SIGTRAP signal; if SIGTRAP is being traced and is not blocked, the lwp stops.

The watchpoint trap occurs before the instruction completes unless WA_TRAPAFTER was specified, in which case it occurs after the instruction completes. If it occurs before completion, the memory is not modified. If it occurs after completion, the memory is modified (if the access is a write access).

pr_info in the lwpstatus structure contains information pertinent to the watchpoint trap. In particular, the si_addr field contains the virtual address of the memory reference that triggered the watchpoint, and the si_code field contains one of TRAP_RWATCH, TRAP_WWATCH, or TRAP_XWATCH, indicating read, write, or execute access, respectively. The si_trapafter field is zero unless WA_TRAPAFTER is in effect for this watched area; non-zero indicates that the current instruction is not the instruction that incurred the watchpoint trap. The si_pc field contains the virtual address of the instruction that incurred the trap.

A watchpoint trap may be triggered while executing a system call that makes reference to the traced process's memory. The lwp that is executing the system call incurs the watchpoint trap while still in the system call. If it stops as a result, the lwpstatus structure contains the system call number and its arguments. If the lwp does not stop, or if it is set running again without clearing the signal or fault, the system call fails with EFAULT. If WA_TRAPAFTER was specified, the memory reference will have completed and the memory will have been modified (if the access was a write access) when the watchpoint trap occurs.

If more than one of WA_READ, WA_WRITE, and WA_EXEC is specified for a watched area, and a single instruction incurs more than one of the specified types, only one is reported when the watchpoint trap occurs. The precedence is WA_EXEC, WA_READ, WA_WRITE (WA_EXEC and WA_READ take precedence over WA_WRITE), unless WA_TRAPAFTER was specified, in which case it is WA_WRITE, WA_READ, WA_EXEC (WA_WRITE takes precedence).

PCWATCH fails with EINVAL if an attempt is made to specify overlapping watched areas or if pr_wflags contains flags other than those specified above. It fails with ENOMEM if an attempt is made to establish more watched areas than the system can support (the system can support thousands).

The child of a vfork(2) borrows the parent's address space. When a vfork(2) is executed by a traced process, all watched areas established for the parent are suspended until the child terminates or performs an exec(2). Any watched areas established independently in the child are cancelled when the parent resumes after the child's termination or exec(2). PCWATCH fails with EBUSY if applied to the parent of a vfork(2) before the child has terminated or performed an exec(2). The PR_VFORKP flag is set in the pstatus structure for such a parent process.

Certain accesses of the traced process's address space by the operating system are immune to watchpoints. The initial construction of a signal stack frame when a signal is delivered to an lwp will not trigger a watchpoint trap even if the new frame covers watched areas of the stack. Once the signal handler is entered, watchpoint traps occur normally. On SPARC based machines, register window overflow and underflow will not trigger watchpoint traps, even if the register window save areas cover watched areas of the stack.

Watched areas are not inherited by child processes, even if the traced process's inherit-on-fork mode, PR_FORK, is set (see PCSET, below). All watched areas are cancelled when the traced process performs a successful exec(2).

PCSET PCUNSET

PCSET sets one or more modes of operation for the traced process. PCUNSET unsets these modes. The modes to be set or unset are specified by flags in an operand long in the control message:

PR_FORK
: (inherit-on-fork): When set, the process's tracing flags and its inherit-on-fork mode are inherited by the child of a fork(2), fork1(2), or vfork(2). When unset, 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 lwps is

closed, all of the process's tracing flags and watched areas are cleared, any outstanding stop directives are canceled, and if any lwps are stopped on events of interest, they are set running as though PCRUN had been applied to them. When unset, the process's tracing flags and watched areas are retained and lwps 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 lwps 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 unset and an lwp stops on an event of interest other than PR_REQUESTED, all other lwps in the process are directed to stop.

PR_MSACCT
: (microstate accounting): When set, microstate accounting is enabled for the process. This allows the usage file to contain accurate values for the times the lwps spent in their various processing states. When unset (the default), the overhead of microstate accounting is avoided and the usage file can only contain an estimate of times spent in the various states.

PR_MSFORK
: (inherit microstate accounting): When set, and microstate accounting is enabled for the process, microstate accounting will be enabled for future child processes. When unset, child processes start with microstate accounting disabled.

PR_BPTADJ
: (breakpoint trap pc adjustment): On IA based machines, a breakpoint trap leaves the program counter (the EIP) referring to the breakpointed instruction plus one byte. When PR_BPTADJ is set, the system will adjust the program counter back to the location of the breakpointed instruction when the lwp stops on a breakpoint. This flag has no effect on SPARC based machines, where breakpoint traps leave the program counter referring to the breakpointed instruction.

PR_PTRACE
: (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 process dies when its parent dies. This mode is deprecated; it is provided only to

allow ptrace(2) 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 pr_flags field of /proc/*pid*/status and /proc/*pid*/lwp/*lwp*/lwpstatus.

PCSREG Set the general registers for the specific or representative lwp according to the operand prgregset_t structure.

On SPARC based systems, only the condition-code bits of the processor-status register (R_PSR) of SPARC V8 (32-bit) processes can be modified by PCSREG. Other privileged registers cannot be modified at all.

On IA based systems, only certain bits of the flags register (EFL) can be modified by PCSREG: these include the condition codes, direction-bit, and overflow-bit.

PCSREG fails with EBUSY if the lwp is not stopped on an event of interest.

PCSVADDR Set the address at which execution will resume for the specific or representative lwp from the operand long. On SPARC based systems, both %pc and %npc are set, with %npc set to the instruction following the virtual address. On IA based systems, only %eip is set. PCSVADDR fails with EBUSY if the lwp is not stopped on an event of interest.

PCSFPREG Set the floating-point registers for the specific or representative lwp according to the operand prfpregset_t structure. 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). PCSFPREG fails with EBUSY if the lwp is not stopped on an event of interest.

PCSXREG Set the extra state registers for the specific or representative lwp according to the architecture-dependent operand prxregset_t structure. An error (EINVAL) is returned if the system does not support extra state registers. PCSXREG fails with EBUSY if the lwp is not stopped on an event of interest.

PCSASRS Set the ancillary state registers for the specific or representative lwp according to the SPARC V9 platform-dependent operand asrset_t structure. An error (EINVAL) is returned if either the target process or the controlling process is not a 64-bit SPARC V9 process. Most of the ancillary state registers are privileged registers that cannot be modified. Only those that can be modified are set; all others are silently ignored. PCSASRS fails with EBUSY if the lwp is not stopped on an event of interest.

PCAGENT Create an agent lwp in the controlled process with register values from the operand prgregset_t structure (see PCSREG, above). The agent lwp is created in the stopped state showing PR_REQUESTED and with its held signal set (the signal mask) having all signals except SIGKILL and SIGSTOP blocked.

The PCAGENT operation fails with EBUSY unless the process is fully stopped via /proc, that is, unless all of the lwps in the process are stopped either on events of interest or on PR_SUSPENDED, or are stopped on PR_JOBCONTROL and have been directed to stop via PCDSTOP. It fails with EBUSY if an agent lwp already exists. It fails with ENOMEM if system resources for creating new lwps have been exhausted.

Any PCRUN operation applied to the process control file or to the control file of an lwp other than the agent lwp fails with EBUSY as long as the agent lwp exists. The agent lwp must be caused to terminate by executing the _lwp_exit(2) system call before the process can be restarted.

Once the agent lwp is created, its lwp-ID can be found by reading the process status file. To facilitate opening the agent lwp's control and status files, the directory name /propc/*pid*/lwp/agent is accepted for lookup operations as an invisible alias for /proc/*pid*/lwp/*lwpid*, *lwpid* being the lwp-ID of the agent lwp (invisible in the sense that the name "agent" does not appear in a directory listing of /proc/*pid*/lwp obtained from ls(1), getdents(2), or readdir(3C)).

The purpose of the agent lwp is to perform operations in the controlled process on behalf of the controlling process: to gather information not directly available via /proc files, or in general to make the process change state in ways not directly available via /proc control operations. To make use of an agent lwp, the controlling process must be capable of making it execute system calls (specifically, the _lwp_exit(2) system call). The register values given to the agent lwp on creation are typically the registers of the representative lwp, so that the agent lwp can use its stack.

The agent lwp is not allowed to execute any variation of the fork(2), exec(2), or _lwp_create(2) system calls. Attempts to do so yield ENOTSUP to the agent lwp.

PCREAD PCWRITE

Read or write the target process's address space via a priovec structure operand:

```
typedef struct priovec {
 void *pio_base;      /* buffer in controlling process */
 size_t pio_len;      /* size of read/write request in bytes */
 off_t pio_offset;    /* virtual address in target process */
} priovec_t;
```

These operations have the same effect as pread(2) and pwrite(2), respectively, of the target process's address space file. The difference is that more than one PCREAD or PCWRITE control operation can be written to the control file at once, and they can be interspersed with other control operations in a single write to the control file. This is useful, for example, when planting many breakpoint instructions in the process's address space, or when stepping over a breakpointed instruction. Unlike `pread`(2) and `pwrite`(2), no provision is made for partial reads or writes; if the operation cannot be performed completely, it fails with `EIO`.

PCNICE

The traced process's `nice`(2) value is incremented by the amount in the operand `long`. Only the super-user may better a process's priority in this way, but any user may lower the priority. This operation is not meaningful for all scheduling classes.

PCSCRED

Set the target process credentials to the values contained in the `prcred_t` structure operand (see `/proc/`*pid*`/cred`). The effective, real, and saved user-IDs and group-IDs of the target process are set. The target process's supplementary groups are not changed; the `pr_ngroups` and `pr_groups` members of the structure operand are ignored. Only the super-user may perform this operation; for all others it fails with `EPERM`.

PROGRAMMING NOTES

For security reasons, except for the `psinfo`, `usage`, `lpsinfo`, `lusage`, `lwpsinfo`, and `lwpusage` files, which are world-readable, and 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. Except for the world-readable files just mentioned, files corresponding to setuid and setgid processes can be opened only by the super-user.

Even if held by the super-user, an open process or lwp file descriptor (other than file descriptors for the world-readable files) becomes invalid if the traced process performs an `exec`(2) of a setuid / setgid object file or an object file that the traced process 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 `PCSET`). This enables a controlling process (if it has permission) to reopen the `/proc` files to get new valid file descriptors, close the invalid file descriptors, unset the run-on-last-close flag (if desired), and proceed. Just closing the invalid file descriptors causes the traced process to resume execution with all tracing flags cleared. Any process not currently open for writing via `/proc`, but that has left-over tracing flags from a previous open, and that executes 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 lwps to stop or terminate, `/proc` file descriptors (other than those obtained by opening the `cwd` or `root` directories or by opening files in the `fd` or `object` directories) can be used in a `poll`(2) system call. When requested and returned, either of the polling events `POLLPRI` or `POLLWRNORM` 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 or POLLWRNORM is requested on a file descriptor referring to a system process (see PCSTOP). The requested events may be empty to wait simply for termination.

FILES

`/proc`	directory (list of processes)
`/proc/`*pid*	specific process directory
`/proc/self`	alias for a process's own directory
`/proc/`*pid*`/as`	address space file
`/proc/`*pid*`/ctl`	process control file
`/proc/`*pid*`/status`	process status
`/proc/`*pid*`/lstatus`	array of lwp status structs
`/proc/`*pid*`/psinfo`	process `ps`(1) info
`/proc/`*pid*`/lpsinfo`	array of lwp `ps`(1) info structs
`/proc/`*pid*`/map`	address space map
`/proc/`*pid*`/rmap`	reserved address map
`/proc/`*pid*`/cred`	process credentials
`/proc/`*pid*`/sigact`	process signal actions
`/proc/`*pid*`/auxv`	process aux vector
`/proc/`*pid*`/ldt`	process LDT (IA only)
`/proc/`*pid*`/usage`	process usage
`/proc/`*pid*`/lusage`	array of lwp usage structs
`/proc/`*pid*`/pagedata`	process page data
`/proc/`*pid*`/watch`	active watchpoints
`/proc/`*pid*`/cwd`	symlink to the current working directory
`/proc/`*pid*`/root`	symlink to the root directory
`/proc/`*pid*`/fd`	directory (list of open files)
`/proc/`*pid*`/fd/*`	aliases for process's open files
`/proc/`*pid*`/object`	directory (list of mapped files)

/proc/*pid*/object/a.out	alias for process's executable file
/proc/*pid*/object/*	aliases for other mapped files
/proc/*pid*/lwp	directory (list of lwps)
/proc/*pid*/lwp/*lwpid*	specific lwp directory
/proc/*pid*/lwp/agent	alias for the agent lwp directory
/proc/*pid*/lwp/*lwpid*/lwpctl	lwp control file
/proc/*pid*/lwp/*lwpid*/lwpstatus	lwp status
/proc/*pid*/lwp/*lwpid*/lwpsinfo	lwp ps(1) info
/proc/*pid*/lwp/*lwpid*/lwpusage	lwp usage
/proc/*pid*/lwp/*lwpid*/gwindows	register windows (SPARC only)
/proc/*pid*/lwp/*lwpid*/xregs	extra state registers
/proc/*pid*/lwp/*lwpid*/asrs	ancillary state registers (SPARC V9 only)

SEE ALSO

ls(1), ps(1), chroot(1M), _lwp_create(2), _lwp_exit(2), alarm(2), brk(2), chdir(2), chroot(2), close(2), creat(2), dup(2), exec(2), fcntl(2), fork(2), fork1(2), fstat(2), getdents(2), kill(2), lseek(2), mmap(2), nice(2), open(2), poll(2), pread(2), ptrace(2), pwrite(2), read(2), readlink(2), readv(2), shmget(2), sigaction(2), sigaltstack(2), vfork(2), wait(2), write(2), writev(2), readdir(3C), siginfo(3HEAD), signal(3HEAD), types32(3HEAD), ucontext(3HEAD)

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	A write(2) was attempted at an illegal address in the traced process.
EBUSY	PCSTOP, PCDSTOP, PCWSTOP, or PCTWSTOP was applied to a system process; an exclusive open(2) was attempted on a /proc file for a process already open for writing; PCRUN, PCSREG, PCSVADDR, PCSFPREG, or PCSXREG was applied to a process or lwp not stopped on an event of interest; an attempt was made to mount /proc when it was already mounted; PCAGENT was applied to a process that was not fully stopped or that already had an agent lwp.

EPERM — Someone other than the super-user issued the PCSCRED operation; someone other than the super-user attempted to better a process's priority by applying PCNICE.

ENOSYS — An attempt was made to perform an unsupported operation (such as creat(2), link(2), or unlink(2)) on an entry in /proc.

EINVAL — In general, this means that some invalid argument was supplied to a system call. A non-exhaustive list of conditions eliciting this error includes: a control message operation code is undefined; an out-of-range signal number was specified with PCSSIG, PCKILL, or PCUNKILL; SIGKILL was specified with PCUNKILL; PCSFPREG was applied on a system that does not support floating-point operations; PCSXREG was applied on a system that does not support extra state registers.

ENOMEM — The system-imposed limit on the number of page data file descriptors was reached on an open of /proc/*pid*/pagedata; an attempt was made with PCWATCH to establish more watched areas than the system can support; the PCAGENT operation was issued when the system was out of resources for creating lwps.

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 PCSTOP, PCWSTOP, or PCTWSTOP.

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.

EOVERFLOW — A 32-bit controlling process attempted to read or write the as file or attempted to read the map, rmap, or pagedata file of a 64-bit target process. A 32-bit controlling process attempted to apply one of the control operations PCSREG, PCSXREG, PCSVADDR, PCWATCH, PCAGENT, PCREAD, PCWRITE to a 64-bit target process.

NOTES

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 `<procfs.h>`.

BUGS

Because the old `ioctl`(2)-based version of `/proc` is currently supported for binary compatibility with old applications, the top-level directory for a process, `/proc/`*pid*, is not world-readable, but it is world-searchable. Thus, anyone can open `/proc/`*pid*`/psinfo` even though `ls`(1) applied to `/proc/`*pid* will fail for anyone but the owner or the super-user. Support for the old `ioctl`(2)-based version of `/proc` will be dropped in a future release, at which time the top-level directory for a process will be made world-readable.

On SPARC based machines, the types `gregset_t` and `fpregset_t` defined in `<sys/regset.h>` are similar to but not the same as the types `prgregset_t` and `prfpregset_t` defined in `<procfs.h>`.

NAME

prof_attr – profile description database

SYNOPSIS

/etc/security/prof_attr

DESCRIPTION

/etc/security/prof_attr is a local source for execution profile names, descriptions, and other attributes of execution profiles. The prof_attr file can be used with other profile sources, including the prof_attr NIS map and NIS+ table. Programs use the getprofattr(3SECDB) routines to gain access to this information.

The search order for multiple prof_attr sources is specified in the /etc/nsswitch.conf file, as described in the nsswitch.conf(4) man page.

An execution profile is a mechanism used to bundle together the commands and authorizations needed to peform a specific function. Each entry in the prof_attr database consists of one line of text containing five fields separated by colons (:). Line continuations using the backslash (\) character are permitted. The format of each entry is:

profname:*res1*:*res2*:*desc*:*attr*

profname	The name of the profile. Profile names are case-sensitive.
res1	Reserved for future use.
res2	Reserved for future use.
desc	A long description. This field should explain the purpose of the profile, including what type of user would be interested in using it. The long description should be suitable for displaying in the help text of an application.
attr	An optional list of semicolon-separated (;) key-value pairs that describe the security attributes to apply to the object upon execution. Zero or more keys may be specified. There are two valid keys, help and auths. help is assigned the name of a file ending in .htm or .html. auths specifies a comma-separated (,) list of authorization names chosen from those names defined in the auth_attr(4) database. Authorization names may be specified using the asterisk (*) character as a wildcard. For example, solaris.printer.* would mean all of Sun's authorizations for printing.

EXAMPLES

EXAMPLE 1 Allowing execution of all commands

The following entry allows the user to execute all commands:

```
All:::Use this profile to give a :help=All.html
```

EXAMPLE 2 Consulting the local prof_attr file first

With the following `nsswitch.conf` entry, the local `prof_attr` file is consulted before the NIS+ table:

```
prof_attr: files nisplus
```

FILES

`/etc/nsswitch.conf`

`/etc/security/prof_attr`

NOTES

When deciding which authorization source to use (see DESCRIPTION), keep in mind that NIS+ provides stronger authentication than NIS.

The root user is usually defined in local databases because root needs to be able to log in and do system maintenance in single-user mode and at other times when the network name service databases are not available. So that the profile definitions for root can be located at such times, root's profiles should be defined in the local `prof_attr` file, and the order shown in the example `nsswitch.conf`(4) file entry under EXAMPLES is highly recommended.

Because the list of legal keys is likely to expand, any code that parses this database must be written to ignore unknown key-value pairs without error. When any new keywords are created, the names should be prefixed with a unique string, such as the company's stock symbol, to avoid potential naming conflicts.

Each application has its own requirements for whether the `help` value must be a relative pathname ending with a filename or the name of a file. The only known requirement is for the name of a file.

The following characters are used in describing the database format and must be escaped with a backslash if used as data: colon (`:`), semicolon (`;`), equals (`=`), and backslash (`\`).

SEE ALSO

`auths`(1), `profiles`(1), `getauthattr`(3SECDB), `getprofattr`(3SECDB), `getuserattr`(3SECDB), `auth_attr`(4), `exec_attr`(4), `user_attr`(4)

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)

OpenWindows 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`(3SOCKET) 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

EXAMPLE 1 A Sample Database

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
egp         8   EGP         # exterior gateway protocol
pup         12  PUP         # PARC universal packet protocol
udp         17  UDP         # user datagram protocol

#
# Internet (IPv6) extension headers
#
hopopt      0   HOPOPT      # Hop-by-hop options for IPv6
ipv6        41  IPv6        # IPv6 in IP encapsulation
ipv6-route  43  IPv6-Route  # Routing header for IPv6
ipv6-frag   44  IPv6-Frag   # Fragment header for IPv6
esp         50  ESP         # Encap Security Payload for IPv6
ah          51  AH          # Authentication Header for IPv6
ipv6-icmp   58  IPv6-ICMP   # IPv6 internet control message protocol
ipv6-nonxt  59  IPv6-NoNxt  # No next header extension header for IPv6
ipv6-opts   60  IPv6-Opts   # Destination Options for IPv6
```

FILES

`/etc/nsswitch.conf` configuration file for name-service switch

SEE ALSO

getprotobyname(3SOCKET), 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, and so forth. 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 that indicates the file type. Valid values are:

- `b` block special device
- `c` character special device
- `d` directory
- `e` a file to be edited upon installation or removal (may be shared by several packages)
- `f` a standard executable or data file
- `i` installation script or information file
- `l` linked file
- `p` named pipe
- `s` symbolic link
- `v` volatile file (one whose contents are expected to change, like a log file)
- `x` an exclusive directory accessible only by this package

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 of the form `$`*variable.* If *variable* begins with a lower case letter, it is a build variable. If *variable* begins with an upper case letter, it is an install variable. Build variables are bound at build time. If an install variable is known at build time, its definition is inserted into the `pkginfo`(4) file so that it will be available at install time. If an install variable is not known at build time, it will be bound at install time.

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.

The mode can be a variable specification of the form `$`*variable.* If *variable* begins with a lower case letter, it is a build variable. If *variable* begins with an upper case letter, it is an install variable. Build variables are bound at build time. If an install variable is known at build time, its definition is inserted into the `pkginfo`(4) file so that it will be available at install time. If an install variable is not known at build time, it will be bound at install time.

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.

The owner can be a variable specification of the form `$`*variable.* If *variable* begins with a lower case letter, it is a build variable. If *variable* begins with an upper case letter, it is an install variable. Build variables are bound at build time. If an install variable is known at build time, its definition is inserted into the `pkginfo`(4) file so that it will be available at install time. If an install variable is not known at build time, it will be bound at install 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.

The group can be a variable specification of the form `$`*variable.* If *variable* begins with a lower case letter, it is a build variable. If *variable* begins with an upper case letter, it is an install variable. Build variables are bound at build time. If an install variable is known at build time, its definition is inserted into the `pkginfo`(4) file so that it will be available at install time. If an install variable is not known at build time, it will be bound at install 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 base name of the *path* field is appended to each directory in the ordered list until the file is located. Searches are not recursive.

`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. Spans to subsequent included prototype files.

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 (that is, *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 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 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)

Application Packaging Developer's Guide

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

EXAMPLE 1 A sample configuration file.

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`(3NSL), `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

EXAMPLE 1 A sample file.

```
#
#
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), crontab(1), nice(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 and Data Communications 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

EXAMPLE 1 The capability continuation feature.

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 and Data Communications Administration Guide

NAME

resolv.conf – configuration file for name server routines

DESCRIPTION

This file helps initialize routines from the `resolver`(3RESOLV) C library. The resolver routines provide access to the Internet Domain Name System.

The resolver configuration file contains information that is read by the resolver routines the first time a process calls them. The file is designed to be human readable and contains a list of keyword-value pairs that provide various types of resolver information. Keyword-value pairs are of the form:

keyword value

The different configuration options are:

`nameserver` *address*

Specifies the Internet address in dot-notation format of one name server to which the resolver should direct any queries. Up to *MAXNS* (currently three) name servers may be listed, on as many as *MAXNS* `nameserver` lines in `resolv.conf`. If multiple servers are specified, the resolver routines query them in the order listed. If no `nameserver` lines are present in the file, resolver routines use the name server on the local machine.

The algorithm of the resolver routines is: try the first name server specified. If the query times out, try the next server listed in the configuration file, and so on until the complement of servers there has been exhausted. If those queries also time out, try the full complement of name servers again, until the maximum number of retry passes has been made.

`domain`*name*

Specifies a local domain name for use as the default domain.

Most queries for names within a domain can use short names relative to the local domain. If a `domain` line is missing from the configuration file, the domain is determined from the environment variable, `LOCALDOMAIN`, if it is defined, from the domain name (see `domainname`(1M)) by omitting the first level, or from the host name (`gethostname`(3C)) by using everything after the first dot. Finally, if the host name does not contain a domain part, the root domain is assumed.

`search`*searchlist*

Specifies a search list for host-name lookup. The search list is normally determined from the local domain name; by default, it contains only the local domain name. This may be changed by listing the desired domains for searches in *searchlist*. Spaces or tabs must separate domain names.

Most resolver queries are attempted using each component of the search path in turn until a match is found. Note that this process may be slow and will generate a lot of network traffic if the servers for the listed domains are not local. Also queries will time out if no server is available for one of the domains.

The search list is currently limited to six domains with a total of 256 characters.

`sortlist`*addresslist*

Causes addresses returned by `gethostbyname`(3NSL) to be sorted in accordance with local rules. A sortlist is specified by IP address netmask pairs. The netmask is optional and defaults to the natural netmask of the net. The IP address and optional network pairs are separated by slashes. Up to 10 pairs may be specified. For example, the following specification requires `gethostbyname()` to return the netmask pair `130.155.160.0/255.255.240.0` ahead of the IP address `130.155.0.0`.

```
sortlist
130.155.160.0/255.255.240.0
130.155.0.0
```

`options`*optionlist*

Specifies optional behaviors for various resolver routines in accordance with *optionlist* values, each of which is equivalent to an internal resolver variable.

The values that may be included as individual *optionlist* values are:

`debug`	Sets `RES_DEBUG` in the `_res.options` field.
`ndots:`*n*	Sets a floor threshold for the number of dots which must appear in a name given to `res_query()` (see `resolver`(3RESOLV)) before an initial absolute (as-is) query is performed. The default for *n* is `1`. Thus, if there are any dots in a name, the name is tried first as an absolute name before any search-list domain names are appended to it.
`retry:`*n*	Sets the number of attempts made to connect to each name server. While `retry:0` is allowed, it is equivalent to `retry:1`. The default is 4.
`retrans:`*n*	Sets the basic retransmit timeout, in seconds. The default is 5. An exponential backoff algorithm is used, so the default values for retry and retrans result in 5+10+20+40=75 seconds of total timeout for each name server. While `retrans:0` is allowed, it is equivalent to `retrans:1`.

The `domain` and `search` keywords are mutually exclusive. If more than one instance of these keywords is present, the last instance takes precedence.

The options established through any `search` lines in the local `resolv.conf` file can be overridden on a per-process basis by setting the environment variable, LOCALDOMAIN, to a space-separated list of search domains.

The options established through any `options` lines in the local `resolv.conf` file can be amended on a per-process basis by setting the environment variable, RES_OPTIONS, to a space-separated list of resolver options, These options are listed above under the `options` keyword.

The keyword-value pair must appear on a single line, and the keyword (for instance, `nameserver`) must start the line. The value or value list follows the keyword, separated from it by white space characters.

FILES

`/etc/resolv.conf`

SEE ALSO

`domainname`(1M), `in.named`(1M), `gethostbyname`(3NSL), `gethostname`(3C), `resolver`(3RESOLV)

Vixie, Paul, Dunlap, Keven J., Karels, Michael J., *Name Server Operations Guide for BIND* (public domain), Internet Software Consortium, 1996.

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. It can also direct the `rmmount` utility to run `fsck` on one or more file systems before mounting them, with the `fsck` command line options specified in `rmmount.conf`.

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 a medium 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 medium, 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

# Mount command options
mount media_or_file_system [file_system_spec] -o mount_command_options

# Optionally fsck command options
fsck media_type filesystem_type -o fsck_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 medium 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 medium. 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 medium (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.

Explanations of the syntax for the `Mount command options` fields are as follows:

media_or_file_system — Either the type of medium (CD-ROM or floppy) or the specific file system to share.

file_system_spec — Specifies one or more file systems to which this line applies. Defaults to "all" file system types.

mount_command_options — One or more options to be passed to the `mount` command. Multiple options require a space delimiter.

Explanations of the syntax for the `fsck` command options fields are as follows:

media_type — The type of removable medium. A Bourne shell regular expression that matches names of file system media whose aliases are listed under `/vol/dev/aliases`. Examples include `cdrom0`, `cdrom1`, `cdrom*`, `floppy0`, and `floppy1`, and `floppy*`.

filesystem_type — The type of file system, for example, `ufs` or `hsfs`, that resides on the medium specified in *media_type*.

fsck_command_options — One or more options to be passed to `fsck`(1M). Multiple options must be separated by spaces.

The algorithm for the `fsck` configuration line is as follows:

1. The `fsck` configuration line tells `rmmount` to run `fsck` on *filesystem_type*, as described above. The *filesystem_type* must be correct for the *media_type* specified.
2. If *filesystem_type* is not present, `rmmount` runs `fsck` on all file systems on all media that match *media_type*.
3. If `rmmount.conf` contains no `fsck` configuration line or contains an `fsck` configuration line with a *media_type* that does not match a medium's alias, `rmmount` does not run `fsck` on the removable medium's file system, unless `mount` reports that the file system's dirty bit is set.

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 rmscsi pcmem
ident pcfs ident_pcfs.so floppy rmscsi pcmem
ident udfs ident_udfs.so cdrom floppy

# Actions
action cdrom action_filemgr.so
action floppy action_filemgr.so
action rmscsi action_filemgr.so
```

EXAMPLES

EXAMPLE 1 Sharing of various file systems.

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" netgroup.

`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.

EXAMPLE 2 Customizing mount operations

The following examples show how different `mount` options could be used to customize how `rmmount` mounts various media:

`mount cdrom* hsfs -o nrr`
mounts all High Sierra CD-ROMs with the `nrr` (no Rock Ridge extensions) option (see `mount_hsfs`(1M))

`mount floppy1 -o ro`
will always mount the second floppy disk read-only (for all file system types)

`mount floppy1 -o ro foldcase`
will always mount the second floppy disk read-only (for all file system types) and pass the `foldcase` mount option

EXAMPLE 3 Telling rmmount to check file systems before mounting them

The following examples show how to tell `rmmount` to check file systems with `fsck` before mounting them, and how to specify the command line options to be used with `fsck`.

`fsck floppy* ufs –o f`
Performs a full file system check on any UFS floppies, ignoring the clean flag, before mounting them.

`fsck floppy* ufs –o p`
Uses the `fsck` `p` (preen) flag for all UFS floppies.

`fsck cdrom* –o f`
Tells `rmmount` to run `fsck` before mounting any file system on CD-ROM.

SEE ALSO

`volcancel`(1), `volcheck`(1), `volmissing`(1), `mount`(1M), `mount_hsfs`(1M), `rmmount`(1M), `share`(1M), `vold`(1M), `vold.conf`(4), `volfs`(7FS)

NOTES

When using the `mount` options line, verify that the specified options will work with the specified file system types. The `mount` command will fail if an incorrect mount option / file system combination is specified. Multiple `mount` options require a space delimiter.

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

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

EXAMPLE 1 RPC Database

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`

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`

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	IA

SEE ALSO

`rpld`(1M), `attributes`(5)

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

EXAMPLE 1 A sample dispadmin configuration file.

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)

System Administration Guide, Volume 1

System Interface Guide

NAME

sbus – configuration files for SBus device drivers

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()`.

  You can use the `ddi_get*` and `ddi_put*` family of functions to access register space from a high-level interrupt context.

`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()`.

`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` 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

EXAMPLE 1 A sample configuration file.

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;
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	SPARC

SEE ALSO

`driver.conf`(4), `attributes`(5), `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 `^Ah` provides a *magic number* of (octal) 064001.

Delta Table

The delta table consists of a variable number of entries of the form:

`^As` *inserted* / *deleted* / *unchanged*

`^Ad` *type sid yr* / *mo* / *da hr* : *mi* : *se username serial-number predecessor-sn*

`^Ai` *include-list*

`^Ax` *exclude-list*

`^Ag` *ignored-list*

`^Am` *mr-number*

`. . .`

`^Ac` *comments* `. . .`
`. . .`

```
^Ae
```

The first line (^As) contains the number of lines inserted / deleted / unchanged respectively. The second line (^Ad) 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 ^Ai, ^Ax, and ^Ag lines contain the serial numbers of deltas included, excluded, and ignored, respectively. These lines do not always appear.

The ^Am lines (optional) each contain one MR number associated with the delta; the ^Ac lines contain comments associated with the delta.

The ^Ae 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 ^Au and ^AU. 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:

```
^Af flag
        optional text
```

The following flags are defined in order of appearance:

^Af t *type-of-program*
  Defines the replacement for the `17:21:50` ID keyword.

^Af 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.

^Af 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.

^Af b
  Indicates that the –`b` option may be used with the SCCS `get` command to create a branch in the delta tree.

^Af m module name
  Defines the first choice for the replacement text of the `sccsfile.4` 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 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:

```
^AI ddddd
^AD ddddd
^AE ddddd
```

respectively. The digit string is the serial number corresponding to the delta for the control line.

SEE ALSO

`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), `sccs`(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`(7D) and `st`(7D) 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_init_pkt`(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` recognize 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`.

The SCSI target driver configuration files shipped with Solaris have entries for LUN `0` only. For devices that support other LUNs, such as some CD changers, the system administrator may edit the driver configuration file to add entries for other LUNs.

EXAMPLES

EXAMPLE 1 A sample configuration file.

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;
```

Add the following lines to `sd.conf` for a six- CD changer on `target 3`, with LUNs `0` to `5`.

```
name="sd" class="scsi" target=3 lun=1;
name="sd" class="scsi" target=3 lun=2;
```

```
name="sd" class="scsi" target=3 lun=3;
name="sd" class="scsi" target=3 lun=4;
name="sd" class="scsi" target=3 lun=5;
```

It is not necessary to add the line for LUN `0`, as it already exists in the file shipped with Solaris.

SEE ALSO

`driver.conf`(4), `sd`(7D), `st`(7D), `scsi_ifgetcap`(9F), `scsi_init_pkt`(9F), `scsi_transport`(9F)

Writing Device Drivers

ANSI Small Computer System Interface-2 (SCSI-2)

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, 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.

NAME

securenets – configuration file for NIS security

SYNOPSIS

`/var/yp/securenets`

DESCRIPTION

The `/var/yp/securenets` file defines the networks or hosts which are allowed access to information by the Network Information Service (“NIS”).

The format of the file is as follows:

- Lines beginning with the “#” character are treated as comments.
- Otherwise, each line contains two fields separated by white space. The first field is a netmask, the second a network.
- The netmask field may be either `255.255.255.255` (IPv4), `ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff` (IPv6) , or the string “host” indicating that the second field is a specific host to be allowed access.

Both `ypserv`(1M) and `ypxfrd`(1M) use the `/var/yp/securenets` file. The file is read when the `ypserv`(1M) and `ypxfrd`(1M) daemons begin. If `/var/yp/securenets` is present, `ypserv`(1M) and `ypxfrd`(1M) respond only to IP addresses in the range given. In order for a change in the `/var/yp/securenets` file to take effect, you must kill and restart any active daemons using `ypstop`(1M) and `ypstart`(1M).

EXAMPLES

EXAMPLE 1 Access for Individual Entries

If individual machines are to be give access, the entry could be:

```
255.255.255.255 192.9.1.20
```

or

```
host 192.0.1.20
```

EXAMPLE 2 Access for a Class C Network

If access is to be given to an entire class C network, the entry could be:

```
255.255.255.0 192.9.1.0
```

EXAMPLE 3 Access for a Class B Network

The entry for access to a class B network could be:

```
255.255.0.0 9.9.0.0
```

EXAMPLE 4 Access for an Invidual IPv6 Address

Similarly, to allow access for an individual IPv6 address:

```
ffff:ffff:ffff:ffff:ffff:ffff:ffff:ffff  fec0::111:abba:ace0:fba5e:1
```

or

```
host  fec0::111:abba:ace0:fba5e:1
```

EXAMPLE 5 Access for all IPv6 Addresses Starting with fe80

To allow access for all IPv6 addresses starting with fe80:

```
ffff::  fe80::
```

FILES

`/var/yp/securenets` Configuration file for NIS security.

SEE ALSO

`ypserv`(1M), `ypstart`(1M), `ypstop`(1M), `ypxfrd`(1M)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

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`(3SOCKET) 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`(3SOCKET), `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`	shadow password file
`/etc/passwd`	password file
`/etc/nsswitch.conf`	name-service switch configuration file

SEE ALSO

`login`(1), `passwd`(1), `useradd`(1M), `userdel`(1M), `usermod`(1M), `getspnam`(3C), `putspent`(3C), `nsswitch.conf`(4), `passwd`(4)

NOTES

If password aging is turned on in any name service the *passwd:* line in the `/etc/nsswitch.conf` file must have a format specified in the `nsswitch.conf`(4) man page.

If the `/etc/nsswitch.conf` passwd policy is not in one of the supported formats, logins will not be allowed upon password expiration because the software does not know how to handle password updates under these conditions. See `nsswitch.conf`(4) for additional information.

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

shells – shell database

SYNOPSIS

`/etc/shells`

DESCRIPTION

The `shells` file contains a list of the shells on the system. Applications use this file to determine whether a shell is valid. See `getusershell`(3C). For each shell a single line should be present, consisting of the shell's path, relative to root.

A hash mark (#) indicates the beginning of a comment; subsequent characters up to the end of the line are not interpreted by the routines which search the file. Blank lines are also ignored.

The following default shells are used by utilities: `/bin/bash`, `/bin/csh`, `/bin/jsh`, `/bin/ksh`, `/bin/pfcsh`, `/bin/pfksh`, `/bin/pfsh`, `/bin/sh`, `/bin/tcsh`, `/bin/zsh`, `/sbin/jsh`, `/sbin/sh`, `/usr/bin/bash`, `/usr/bin/csh`, `/usr/bin/jsh`, `/usr/bin/ksh`, `/usr/bin/pfcsh`, `/usr/bin/pfksh`, `/usr/bin/pfsh`, and `/usr/bin/sh`, `/usr/bin/tcsh`, `/usr/bin/zsh`.

FILES

`/etc/shells`	lists shells on system

SEE ALSO

`vipw`(1B), `ftpd`(1M), `sendmail`(1M), `getusershell`(3C), `aliases`(4)

NAME

slp.conf – configuration file for Service Location Protocol agents

SYNOPSIS

`/etc/inet/slp.conf`

DESCRIPTION

`slp.conf` provides all Service Location Protocol (“SLP”) agents with their operational configuration. `slpd`(1M) reads `slp.conf` on startup. Service Agents (“SAs”) and User Agents (“UAs”) read `slp.conf` on invocation of the SA and UA library routines; configuration parameters are then cached on a per-process basis. All SA’s must use the same set of properties as `slpd` on the local machine, since `slpd` acts as an SA server.

The configuration file format consists of a newline-delimited list of zero or more property definitions. Each property definition corresponds to a particular configurable SLP, network, or other parameter in one or more of the three SLP agents. The file format grammar is shown in *RFC 2234* as follows:

```
config-file   =  line-list
line-list     =  line / line line-list
line          =  property-line / comment-line
comment-line  =  ( "#" / ";" ) 1*allchar newline
property-line =  property newline
property      =  tag "=" value-list
tag           =  prop / prop "." tag
prop          =  1*tagchar
value-list    =  value / value "," value-list
value         =  int / bool /
                 "(" value-list ")" / string
int           =  1*DIGIT
bool          =  "true" / "false" / "TRUE" / "FALSE"
newline       =  CR / ( CRLF )
string        =  1*stringchar
tagchar       =  DIGIT / ALPHA / tother / escape
tother        =  %x21-%x2d / %x2f /
                 %x3a / %x3c-%x40 /
                 %x5b-%x60 / %7b-%7e
                 ; i.e., all characters except ‘.’,
                 ; and ‘=’.
stringchar    =  DIGIT / ALPHA / sother / escape
sother        =  %x21-%x29 / %x2a-%x2b /
                 %x2d-%x2f / %x3a-%x40 /
                 %x5b-%x60 / %7b-%7e
                 ; i.e., all characters except ‘,’
allchar       =  DIGIT / ALPHA / HTAB / SP
escape        =  "\" HEXDIG HEXDIG
                 ; Used for reserved characters
```

The properties fall into one of the following categories:

- DA Configuration

- Static Scope Configuration
- Tracing and Logging
- Serialized Proxy Registrations
- Networking Configuration Parameters
- UA Configuration

DA Configuration

The following are configuration properties and their parameters for DAs:

`net.slp.isDA`

Setting Type	Boolean
Default Value	`False`
Range of Values	`True` or `False`

A boolean that indicates whether `slpd`(1M) is to act as a DA. If `False`, `slpd`(1M) is not run as a DA.

`net.slp.DAHeartBeat`

Setting Type	Integer
Default Value	10800 seconds (3 hours)
Range of Values	2000 – 259200000 seconds

A 32–bit integer giving the number of seconds for the passive DA advertisement heartbeat. The default value is 10800 seconds. This property is ignored if `net.slp.isDA` is `False`.

`net.slp.DAAttributes`

Setting Type	List of Strings
Default Value	Unassigned
Range of Values	List of Attribute Tag/Value List Pairs

A comma-separated list of parenthesized attribute tag/value list pairs that the DA must advertise in DA advertisements. The property must be in the SLP attribute list wire format, which requires that you use a backslash ("\") to escape reserved characters. See *RFC 2608* for more information on reserved characters, or refer to the *Service Location Protocol Administration Guide*.

Static Scope Configuration

The following properties and their parameters allow you to configure various aspects of scope and DA handling:

`net.slp.useScopes`

Setting Type	List of Strings
Default Value	`Default`, for SA and DA; unassigned for UA.
Range of Values	List of Strings

A list of strings indicating either the scopes that a UA or an SA is allowed to use when making requests, or the scopes a DA must support. If not present for the DA and SA, the default scope `Default` is used. If not present for the UA, then the user scoping model is in force, in which active and passive DA or SA discovery are used for scope discovery. The scope `Default` is used if no other information is available. If a DA or SA gets another scope in a request, a `SCOPE_NOT_SUPPORTED` error is returned, unless the request was multicast, in which case it is dropped. If a DA receives another scope in a registration, a `SCOPE_NOT_SUPPORTED` error will be returned. Unlike other properties, this property is "read-only", so attempts to change it programmatically after the configuration file has been read are ignored.

`net.slp.DAAddresses`

Setting Type	List of Strings
Default Value	Unassigned
Range of Values	IPv4 addresses or host names

A list of IP addresses or DNS-resolvable names that denote the DAs to use for statically configured UAs and SAs. The property is read by `slpd`(1M), and registrations are forwarded to the DAs. The DAs are provided to UAs upon request. Unlike other properties, this property is "read-only", so attempts to change it after the configuration file has been read are ignored.

The following grammar describes the property:

```
addr-list     =  addr / addr "," addr-list
addr          =  fqdn / hostnumber
fqdn          =  ALPHA / ALPHA *[ anum / "-" ] anum
anum          =  ALPHA / DIGIT
hostnumber    =  1*3DIGIT 3("." 1*3DIGIT)
```

The following is an example using this grammar:

```
sawah,mandi,sambal
```

IP addresses can be used instead of host names in networks where DNS is not deployed, but network administrators are reminded that using IP addresses will complicate machine renumbering, since the SLP configuration property files in statically configured networks will have to be changed.

Tracing and Logging

These properties direct tracing and logging information to be sent to `syslogd` at the `LOG_INFO` priority. These properties affect `slpd`(1M) only.

`net.slp.traceDATraffic`

Setting Type	Boolean
Default Value	`False`
Range of Values	`True` or `False`

Set `net.slp.traceDATraffic` to `True` to enable logging of DA traffic by `slpd`.

`net.slp.traceMsg`

Setting Type	Boolean
Default Value	`False`
Range of Values	`True` or `False`

Set `net.slp.traceMsg` to `True` to display details about SLP messages. The fields in all incoming messages and outgoing replies are printed by `slpd`.

`net.slp.traceDrop`

Setting Type	Boolean
Default Value	`False`
Range of Values	`True` or `False`

Set this property to `True` to display details when an SLPmessage is dropped by `slpd` for any reason.

`net.slp.traceReg`

Setting Type	Boolean
Default Value	`False`
Range of Values	`True` or `False`

Set this property to `True` to display the table of service advertisements when a registration or deregistration is processed by `slpd`.

Serialized Proxy Registrations

The following properties control reading and writing serialized registrations.

`net.slp.serializedRegURL`

Setting Type	String
Default Value	Unassigned
Range of Values	Valid URL

A string containing a URL pointing to a document, which contains serialized registrations that should be processed when the `slpd` starts up.

Networking Configuration Parameters

The properties that follow allow you to set various network configuration parameters:

`net.slp.isBroadcastOnly`

Setting Type	Boolean
Default Value	`False`
Range of Values	`True` or `False`

A boolean that indicates if broadcast should be used instead of multicast.

`net.slp.multicastTTL`

Setting Type	Positive Integer
Default Value	`255`
Range of Values	A positive integer from 1 to 255.

A positive integer less than or equal to 255 that defines the multicast TTL.

`net.slp.DAActiveDiscoveryInterval`

Setting Type	Integer
Default Value	900 seconds (15 minutes)
Range of Values	From 300 to 10800 seconds

A 16–bit positive integer giving the number of seconds between DA active discovery queries. The default value is 900 seconds (15 minutes). If the property is set to zero, active discovery is turned off. This is useful when the DAs available are explicitly restricted to those obtained from the `net.slp.DAAddresses` property.

`net.slp.multicastMaximumWait`

Setting Type	Integer
Default Value	15000 milliseconds (15 seconds)
Range of Values	1000 to 60000 milliseconds

A 32–bit integer giving the maximum value for the sum of the `net.slp.multicastTimeouts` values and `net.slp.DADiscoveryTimeouts` values in milliseconds.

`net.slp.multicastTimeouts`

Setting Type	List of Integers
Default Value	`3000,3000,3000,3000`
Range of Values	List of Positive Integers

A list of 32–bit integers used as timeouts, in milliseconds, to implement the multicast convergence algorithm. Each value specifies the time to wait before sending the next request, or until nothing new has been learned from two successive requests. In a fast network the aggressive values of `1000,1250,1500,2000,4000` allow better performance. The sum of the list must equal `net.slp.multicastMaximumWait`.

`net.slp.passiveDADetection`

Setting Type	Boolean
Default Value	`True`
Range of Values	`True` or `False`

A boolean indicating whether `slpd` should perform passive DA detection.

`net.slp.DADiscoveryTimeouts`

Setting Type	List of Integers.
Default Value	`2000,2000,2000,2000,3000,4000`
Range of Values	List of Positive Integers

A list of 32–bit integers used as timeouts, in milliseconds, to implement the multicast convergence algorithm during active DA discovery. Each value specifies the time to wait before sending the next request, or until nothing new has been learned from two successive requests. The sum of the list must equal `net.slp.multicastMaximumWait`.

`net.slp.datagramTimeouts`

Setting Type	List of Integers
Default Value	`3000,3000,3000`
Range of Values	List of Positive Integers

A list of 32–bit integers used as timeouts, in milliseconds, to implement unicast datagram transmission to DAs. The *n*th value gives the time to block waiting for a reply on the *n*th try to contact the DA.

`net.slp.randomWaitBound`

Setting Type	Integer
Default Value	1000 milliseconds (1 second)
Range of Values	1000 to 3000 milliseconds

Sets the upper bound for calculating the random wait time before attempting to contact a DA.

`net.slp.MTU`

Setting Type	Integer
Default Value	1400
Range of Values	128 to 8192

A 16–bit integer that specifies the network packet size, in bytes. The packet size includes IP and TCP or UDP headers.

`net.slp.interfaces`

Setting Type	List of Strings
Default Value	Default interface
Range of Values	IPv4 addresses or host names

List of strings giving the IP addresses or host names of the network interface cards on which the DA or SA should listen on port 427 for multicast, unicast

UDP, and TCP messages. The default value is unassigned, indicating that the default network interface card should be used. An example is:

```
195.42.42.42,195.42.142.1,195.42.120.1
```

The example machine has three interfaces on which the DA should listen. Note that if IP addresses are used, the property must be renumbered if the network is renumbered.

UA Configuration

The following configuration parameters apply to the UA:

`net.slp.locale`

Setting Type	String
Default Value	`en`
Range of Values	See *RFC 1766* for a list of the locale language tag names.

A *RFC 1766* Language Tag for the language locale. Setting this property causes the property value to become the default locale for SLP messages.

`net.slp.maxResults`

Setting Type	Integer
Default Value	`-1`
Range of Values	`–1`, positive integer

A 32 bit-integer that specifies the maximum number of results to accumulate and return for a synchronous request before the timeout, or the maximum number of results to return through a callback if the request results are reported asynchronously. Positive integers and `-1` are legal values. If the value of `net.slp.maxResults` is `-1`, all results should be returned.

`net.slp.typeHint`

Setting Type	List of Strings
Default Value	Unassigned
Range of Values	Service type names

A list of service type names. In the absence of any DAs, UAs perform SA discovery to find scopes. If the `net.slp.typeHint` property is set, only SA's advertising types on the list respond. Note that UAs set this property programmatically. It is not typically set in the configuration file. The default is unassigned, meaning do not restrict the type.

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWslpr
CSI	Enabled
Interface Stability	Standard

SEE ALSO

`slpd`(1M), `slpd.reg`(4), `slp_api`(3SLP), `slp`(7P)

Service Location Protocol Administration Guide

Alvestrand, H., *RFC 1766: Tags for the Identification of Languages*, Network Working Group, March 1995.

Crocker, D., Overell, P., *RFC 2234, Augmented BNF for Syntax Specifications: ABNF*, The Internet Society, 1997.

Kempf, J. and Guttman, E., *RFC 2614, An API for Service Location*, The Internet Society, June 1999.

NAME

slpd.reg – serialized registration file for the service location protocol daemon (slpd)

SYNOPSIS

`/etc/inet/slpd.reg`

DESCRIPTION

The serialized registration file contains a group of registrations that `slpd`(1M) registers when it starts. These registrations are primarily for older service programs that do not internally support SLP and cannot be converted. The character format of the registration file is required to be ASCII. To use serialized registrations, set the `net.slp.serializedRegURL` property in `slp.conf`(4) to point at a valid `slpd.reg` file. The syntax of the serialized registration file, in ABNF format (see *RFC 2234*), is as follows:

```
ser-file      =  reg-list
reg-list      =  reg / reg reg-list
reg           =  creg / ser-reg
creg          =  comment-line ser-reg
comment-line  =  ( "#" / ";" ) 1*allchar newline
ser-reg       =  url-props [slist] [attr-list] newline
url-props     =  surl "," lang "," ltime [ "," type ] newline
surl          =  ;The registration's URL. See
                 ; [8] for syntax.
lang          =  1*8ALPHA [ "-" 1*8ALPHA ]
                 ;RFC 1766 Language Tag see [6].
ltime         =  1*5DIGIT
                 ; A positive 16-bit integer
                 ; giving the lifetime
                 ; of the registration.
type          =  ; The service type name, see [7]
                 ; and [8] for syntax.
slist         =  "scopes" "=" scope-list newline
scope-list    =  scope-name / scope-name "," scope-list
scope         =  ; See grammar of [7] for
                 ; scope-name syntax.
attr-list     =  attr-def / attr-def attr-list
attr-def      =  ( attr / keyword ) newline
keyword       =  attr-id
attr          =  attr-id "=" attr-val-list
attr-id       =  ;Attribute id, see [7] for syntax.
attr-val-list =  attr-val / attr-val "," attr-val-list
attr-val      =  ;Attribute value, see [7] for syntax
allchar       =  char / WSP
char          =  DIGIT / ALPHA / other
other         =  %x21-%x2f / %x3a-%x40 /
                 %x5b-%x60 / %7b-%7e
                  ; All printable, nonwhitespace US-ASCII
                  ; characters.
newline       =  CR / ( CRLF )
```

The syntax for attributes and attribute values requires that you use a backslash to escape special characters, in addition to non-ASCII characters, as specified in

RFC 2608. The `slpd` command handles serialized registrations exactly as if they were registered by an SA. In the `url-props` production, the type token is optional. If the type token is present for a service: URL, a warning is signalled, and the type name is ignored. If the maximum lifetime of `65535` seconds is specified, the registration is taken to be permanent, and it is continually refreshed by the DA or SA server until it exits.

Scopes can be included in a registration by including an attribute definition with tag `scopes` followed by a comma-separated list of scope names immediately after the `url-props` production. If the optional `scope-list` is present, the registations are made in the indicated scopes; otherwise, they are registered in the scopes with which the DA or SA server was configured through the `net.slp.useScopes` property. If any conflicts occur between the scope list and the `net.slp.useScopes` property, an error message is issued by way of `syslog`(3C). Refer to information regarding `LOG_INFO` in `syslog`(3C).

Service advertisements are separated by a single blank line. Additionally, the file must end with a single blank line.

EXAMPLES

EXAMPLE 1 Using a Serialized Registration File

The following serialized registration file shows an instance of the service type `foo`, with a lifetime of `65535` seconds, in the `en` locale, with scope `somescope`:

```
# register foo
service:foo://fooserver/foopath,en,65535
scopes=somescope
description=bogus
security=kerberos_v5
location=headquarters

# next registration...
```

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Availability	SUNWslpr
CSI	Enabled
Interface Stability	Standard

SEE ALSO

`slpd`(1M), `slp_api`(3SLP), `syslog`(3C), `slp.conf`(4), `attributes`(5)

Crocker, D. and Overell, P., *RFC 2234, Augmented BNF for Syntax Specifications: ABNF*, The Internet Society, November 1997.

Guttman, E., Perkins, C., Veizades, J., and Day, M., *RFC 2608, Service Location Protocol, Version 2*, The Internet Society, June 1999.

Kempf, J. and Guttman, E., *RFC 2614, An API for Service Location*, The Internet Society, June 1999.

NAME

sock2path – file that maps sockets to transport providers

SYNOPSIS

`/etc/sock2path`

DESCRIPTION

The socket mapping file, `/etc/sock2path`, is a system file that contains the mappings between the `socket`(3SOCKET) call parameters and the transport provider driver. Its format is described on the `soconfig`(1M) manual page.

The `init`(1M) utility uses the `soconfig` utility with the `sock2path` file during the booting sequence.

EXAMPLES

EXAMPLE 1 A Sample `sock2path` File

The following is a sample `sock2path` file:

```
 #   Family   Type  Protocol    Path
         2       2       0      /dev/tcp
         2       2       6      /dev/tcp

         26      2       0      /dev/tcp6
         26      2       6      /dev/tcp6

         2       1       0      /dev/udp
         2       1       17     /dev/udp

         26      1       0      /dev/udp6
         26      1       17     /dev/udp6

         1       2       0     /dev/ticotsord
         1       6       0     /dev/ticotsord
         1       1       0     /dev/ticlts

         2       4       0     /dev/rawip
         26      4       0     /dev/rawip6

         24      4       0     /dev/rts

         27      4       2     /dev/keysock
```

SEE ALSO

`soconfig`(1M), `socket`(3SOCKET)

Network Interfaces Programmer's Guide

NAME

space – disk space requirement file

DESCRIPTION

`space` is an ASCII file that gives information about disk space requirements for the target environment. The `space` file defines space needed beyond what is used by objects defined in the `prototype`(4) file; for example, files which will be installed with the `installf`(1M) command. The `space` file 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

EXAMPLE 1 A sample file.

```
# extra space required by config data which is
# dynamically loaded onto the system
data 500 1
```

SEE ALSO

`installf`(1M), `prototype`(4)

Application Packaging Developer's Guide

NAME

sulog – su command log file

SYNOPSIS

`/var/adm/sulog`

DESCRIPTION

The `sulog` file is a record of all attempts by users on the system to execute the `su`(1M) command. Each time `su`(1M) is executed, an entry is added to the `sulog` file.

Each entry in the `sulog` file is a single line of the form:

```
SU date time
result port user-newuser
```

where

`date` The month and date `su`(1M) was executed. `date` is displayed in the form *mm*/dd where *mm* is the month number and dd is the day number in the month.

`time` The time `su`(1M) was executed. `time` is displayed in the form *HH/MM* where *HH* is the hour number (24 hour system) and *MM* is the minute number.

result The result of the `su`(1M) command. A ‘ + ’ sign is displayed in this field if the su attempt was successful; otherwise a ‘ - ’ sign is displayed.

port The name of the terminal device from which `su`(1M) was executed.

user The user id of the user executing the `su`(1M) command.

newuser The user id being switched to with `su`(1M).

EXAMPLES

EXAMPLE 1 A sample `sulog` file.

Here is a sample `sulog` file:

```
SU 02/25 09:29 + console root-sys
SU 02/25 09:32 + pts/3 user1-root
SU 03/02 08:03 + pts/5 user1-root
SU 03/03 08:19 + pts/5 user1-root
SU 03/09 14:24 - pts/5 guest3-root
SU 03/09 14:24 - pts/5 guest3-root
SU 03/14 08:31 + pts/4 user1-root
```

FILES

`/var/adm/sulog` su log file

`/etc/default/su` contains the default location of `sulog`

SEE ALSO su(1M)

NAME

sysbus, isa, eisa – device tree properties for ISA and EISA bus device drivers

DESCRIPTION

Solaris (Intel Platform Edition) supports the ISA and EISA buses as the system bus. Drivers for devices on these buses use the device tree built by the booting system to retrieve the necessary system resources used by the driver. These resources include 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 and EISA device drivers are only necessary to describe properties used by a particular driver that are not part of the standard properties found in the device tree. See `driver.conf`(4) for further details of configuration file syntax.

The ISA and EISA nexus drivers all belong to class `sysbus` . All bus drivers of class `sysbus` recognize the following properties:

`interrupts`
: An arbitrary-length array where each element of the array represents a hardware interrupt (IRQ) that is used by the device. In general, this array only has one entry unless a particular device uses more than one IRQ.

  Solaris defaults all ISA and EISA interrupts to IPL 5. This interrupt priority may be overridden by placing an `interrupt-priorities` property in a .conf file for the driver. Each entry in the array of integers for the `interrupt-priorities` property is matched one-to-one with the elements in the `interrupts` property to specify the `IPL` value that will be used by the system for this interrupt in this driver. 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. Priorities `11` and greater are high level priorities and are generally not recommended (see `ddi_intr_hilevel`(9F)).

  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 will have an `interrupts` property.

`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 specifies the memory type, `0` specifies a memory range and `1` specifies an I/O range. The second integer specifies the base address of the memory range. 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()` .

All `sysbus` devices will have `reg` properties. The first tuple of this property is used to construct the address part of the device name under `/devices` . In the case of `Plug and Play ISA` devices, the first tuple is a special tuple that does not denote a memory range, but is used by the system only to create the address part of the device name. This special tuple can be recognized by determining if the top bit of the first integer is set to a one.

The order of the tuples in the reg property is determined by the boot system probe code and depends on the characteristics of each particular device. However, the reg property will maintain the same order of entries from system boot to system boot. The recommended way to determine the reg property for a particular device is to use the `prtconf`(1M) command after installing the particular device. The output of the `prtconf` command can be examined to determine the reg property for any installed device.

You can use the `ddi_get*` and `ddi_put*` family of functions to access register space from a high-level interrupt context.

`dma-channels` A list of integers that specifies the DMA channels used by this device. Only devices that use DMA channels will have a `dma-channels` property.

It is recommended that drivers for devices connected to the system bus recognize the following standard property names:

`slot` The number of the slot containing the device, if known. (Only for `EISA` devices).

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Architecture	IA

SEE ALSO

prtconf(1M) , driver.conf(4) , scsi(4) , attributes(5) , ddi_add_intr(9F) , ddi_intr_hilevel(9F) , ddi_map_regs(9F) , ddi_prop_op(9F)

Writing Device Drivers

NAME

sysidcfg – system identification configuration file

DESCRIPTION

When a diskless client boots for the first time or a system installs over the network, the booting software tries to obtain configuration information about the system (such as the system's root password or name service) from a `sysidcfg` file first and then the name service databases. If the booting software cannot find the information, it prompts the user to provide the appropriate information. Like the name service databases, the `sysidcfg` file can be used to avoid all the prompts and provide a totally hands-off booting process.

The `sysidcfg` file preconfigures information through a set of keywords, and you can specify one or more of the keywords to preconfigure as much information as you want. Also, every system that requires different configuration information must have a different `sysidcfg` file. For example, you can use the same `sysidcfg` file to preconfigure the time zone for multiple systems if you want all the systems to have the same time zone configured. However, if you want to preconfigure a different root password for each of those systems, then each system would need its own `sysidcfg` file.

Where To Put the sysidcfg File

The `sysidcfg` file can reside on a shared NFS network directory or the root directory on a UFS or PCFS diskette in the system's diskette drive. If you put the `sysidcfg` file on a shared NFS network directory, you have to use the –`p` option of the `add_install_client`(1M) command (see `install_scripts`(1M)) to specify where the system being installed can find the `sysidcfg` file. If you put the `sysidcfg` file on a diskette, you need to make sure the diskette is in the system's diskette drive when the system boots (on IA systems, the `sysidcfg` file should reside on the Solaris Device Configuration Assistant diskette).

Only one `sysidcfg` file can reside in a directory or diskette. If you are creating more than one `sysidcfg` file, they must reside in different directories or diskettes.

Keyword Syntax Rules

The following rules apply to the keywords in a `sysidcfg` file:

- Keywords can be in any order
- Keywords are not case sensitive
- Keyword values can be optionally enclosed in single (`'`) or double (`"`) quotes
- Only the first instance of a keyword is valid; if you specify the same keyword more than once, the first keyword specified will be used.

Keywords – All Platforms

Name service, domain name, name server

Keywords	Example
name_service=`NIS`, `NIS+`, `DNS`, `NONE`	

Keywords	Example
Options for NIS and NIS+: domain_name=*domain_name;* name_server=*hostname*(*ip_address*)	name_service=NIS {domain_name=west.arp.com name_server=timber(129.221.2.1)}
	name_service=NIS+ {domain_name=west.arp.com name_server=timber(129.221.2.1)}
Options for DNS: domain_name=*domain_name*; name_server=*ip_address*, *ip_address*, *ip_address* (three maximum); search=*domain_name*, *domain_name*, *domain_name*, *domain_name*, *domain_name*, *domain_name* (six maximum, total length less than or equal to 250 characters)	name_service=DNS {domain_name=west.arp.com name_server=10.0.1.10,10.0.1.20 search=arp.com,east.arp.com}

Choose only one value for `name_service`. Include either, both, or neither of the `domain_name` and `name_server` keywords, as needed. If no keywords are used, omit the curly braces {}.

Network interface, host name, Internet Protocol (IP) address, netmask, DHCP, IPv6

Keywords	Example
`network_interface`=`NONE`, `PRIMARY`, *value*	
If DHCP is to be used, the options for `PRIMARY` and *value* are: `dhcp`; `protocol_ipv6`=*yes_or_no*	network_interface=primary {dhcp protocol_ipv6=yes}
If DHCP is not to be used, the options for `PRIMARY` and *value* are: `hostname`=*host_name*; `ip_address`=*ip_address*; `netmask`=*netmask*; `protocol_ipv6`=*yes_or_no*	network_interface=le0 {hostname=feron ip_address=129.222.2.1 netmask=255.255.0.0 protocol_ipv6=no}

Choose only one value for `network_interface`. Include any combination or none of the `hostname`, `ip_address`, and `netmask` keywords, as needed. If you do not use any of these keywords, omit the curly braces {}.

`protocol_ipv6` is optional; you do not need to specify it.

Root password

Keywords	**Values**
`root_password=`*root_password*	Encrypted from `/etc/shadow`

Language in which to display the install program

Keywords	**Values**
`system_locale=`*locale*	`/usr/lib/locale`

Terminal type

Keywords	**Values**
timezone=*timezone*	/usr/share/lib/zoneinfo/*

Date and time

Keywords	Values
`timeserver=`*localhost, hostname, ip_address*	If you specify localhost as the time server, the system's time is assumed to be correct. If you specify the hostname or *ip_address* (if you are not running a name service) of a system, that system's time is used to set the time.

Keywords — IA Platform

Monitor type

Keywords	**Values**
`monitor=`*monitor_type*	Append `kdmconfig –d` *filename* output to `sysidcfg` file

Keyboard language, keyboard layout

Keywords	**Values**
`keyboard=`*keyboard_language* {layout=*value*}	Append `kdmconfig –d` *filename* output to `sysidcfg` file

Graphics card, color depth, display resolution, screen size

Keywords	Values
`display`=*graphics_card* {`size`=*screen_size* `depth`=*color_depth* `resolution`=*screen_resolution*}	Append `kdmconfig –d` *filename* output to `sysidcfg` file

Printing device, number of buttons, IRQ level

Keywords	Values
`pointer`=*pointing_device* {`nbuttons`=*number_buttons* `irq`=*value*}	Append `kdmconfig –d` *filename* output to `sysidcfg` file

EXAMPLES

EXAMPLE 1 Sample `sysidcfg` files

The following example is a `sysidcfg` file for a group of SPARC systems to install over the network. (The host names, IP addresses, and netmask of these systems have been preconfigured by editing the name service.) Because all the system configuration information has been preconfigured, an automated installation can be created by using a custom JumpStart profile.

```
system_locale=en_US
timezone=US/Central
timeserver=localhost
terminal=sun-cmd
name_service=NIS {domain_name=marquee.central.sun.com
                  name_server=connor(129.152.112.3)}
root_password=m4QPOWNY
system_locale=C
```

The following example is a `sysidcfg` file created for a group of IA systems to install over the network that all have the same keyboard, graphics cards, and pointing devices. The device information (keyboard, display, and pointer) was captured from running `kdmconfig –d` (see `kdmconfig`(1M)). In this example, users would see only the prompt to select a language (*system_locale*) for displaying the rest of the Solaris installation program.

```
keyboard=ATKBD {layout=US-English}
display=ati {size=15-inch}
pointer=MS-S
timezone=US/Central
timeserver=connor
terminal=AT386
name_service=NIS {domain_name=marquee.central.sun.com
                  name_server=connor(129.152.112.3)}
root_password=URFUni9
```

SEE ALSO

`install_scripts`(1M), `kdmconfig`(1M), `sysidtool`(1M)

Solaris 8 Advanced Installation Guide

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), among others.

`lpr` The line printer spooling system: `lpr`(1B), `lpc`(1B), among others.

`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), among others.

`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. A configuration entry with a *level* value of `notice` must appear on a separate line.

`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

EXAMPLE 1 A Sample Configuration File

With the following configuration file:

```
*.notice                        /var/log/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`(1M) 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), lpc(1B), lpr(1B), m4(1), cron(1M), getty(1M), in.ftpd(1M), su(1M), syslogd(1M), syslog(3C), hosts(4)

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`

IA 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

EXAMPLE 1 A sample `system` file.

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 configuration option {_POSIX_CHOWN_RESTRICTED} :
* This configuration option is enabled by default.
set rstchown = 1
* Disable the configuration option {_POSIX_CHOWN_RESTRICTED} :
set rstchown = 0
* 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 boot using `boot -a`, which will ask you to specify the path to the saved file. This should allow the system to boot correctly. If you cannot locate a `system` file that will work, you may specify `/dev/null`. This acts as an empty `system` file, and the system will attempt to boot using its default settings.

NOTES

`/etc/system` is only read once; at boot time.

NAME

telnetrc – file for telnet default options

DESCRIPTION

The `.telnetrc` file contains commands that are executed when a connection is established on a per-host basis. Each line in the file contains a host name, one or more spaces or tabs, and a `telnet`(1) command. The host name, `DEFAULT`, matches all hosts. Lines beginning with the pound sign (#) are interpreted as comments and therefore ignored. `telnet`(1) commands are case-insensitive to the contents of the `.telnetrc` file.

The `.telnetrc` file is retrieved from each user's HOME directory.

EXAMPLES

EXAMPLE 1 A sample file.

In the following example, a `.telnetrc` file executes the `telnet`(1) command, `toggle`:

```
weirdhost toggle crmod
# Always export $PRINTER
DEFAULT environ export PRINTER
```

The lines in this file indicate that the `toggle` argument `crmod`, whose default value is "off" (or `FALSE`), should be enabled when connecting to the system `weirdhost`. In addition, the value of the environment variable `PRINTER` should be exported to all systems. In this case, the `DEFAULT` keyword is used in place of the host name.

FILES

`$HOME/.telnetrc`

SEE ALSO

`telnet`(1), `in.telnetd`(1M), `environ`(5)

NAME

term – format of compiled term file

SYNOPSIS

`/usr/share/lib/terminfo/?/*`

DESCRIPTION

The `term` file is compiled from `terminfo`(4) source files using `tic`(1M). Compiled files are organized in a directory hierarchy under the first letter of each terminal name. For example, the `vt100` file would have the pathname `/usr/lib/terminfo/v/vt100`. The default directory is `/usr/share/lib/terminfo`. 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`(3CURSES)). 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 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 name 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 terminal name section is followed by the Boolean section, number section, string section, and string table.

The boolean flags section consists of 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 flags 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,
```

The following is an octal dump of the corresponding `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
/usr/xpg4/include/term.h	X/Open Curses terminfo header

SEE ALSO

infocmp(1M), curses(3CURSES), curses(3XCURSES), terminfo(4), term(5)

NAME

terminfo – terminal and printer capability database

SYNOPSIS

/usr/share/lib/terminfo/?/*

DESCRIPTION

terminfo is a database 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-based 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 descriptions are located in the directory pointed to by the environment variable TERMINFO or in /usr/share/lib/terminfo. terminfo descriptions are generated by tic(1M).

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. Each device entry has the following format:

```
alias1 | alias2 | . . . | aliasn | fullname,
         capability1, capability2,
         .
         .
         .
         capabilityn,
```

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:

- PART 1: DEVICE CAPABILITIES
- PART 2: 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
dest_tabs_magic_smso	xt	xt	Destructive tabs, magic `smso` char (t1061)
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

Variable	Cap-name	Termcap Code	Description
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)
no_pad_char	npc	NP	Pad character doesn't exist
non_dest_scroll_region	ndscr	ND	Scrolling region is nondestructive
non_rev_rmcup	nrrmc	NR	`smcup` does not reverse `rmcup`
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
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
bit_image_entwining	bitwin	Yo	Number of passes for each bit-map row
bit_image_type	bitype	Yp	Type of bit image device
buffer_capacity	bufsz	Ya	Number of bytes buffered before printing
buttons	btns	BT	Number of buttons on the mouse
columns	cols	co	Number of columns in a line

Variable	Cap-name	Termcap Code	Description
dot_horz_spacing	spinh	Yc	Spacing of dots horizontally in dots per inch
dot_vert_spacing	spinv	Yb	Spacing of pins vertically in pins 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
max_attributes	ma	ma	Maximum combined video attributes
			terminal can display
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
maximum_windows	wnum	MW	Maximum number of definable windows
micro_char_size	mcs	Yf	Character step size when in micro mode
micro_line_size	mls	Yg	Line step size when in micro mode

Variable	Cap-name	Termcap Code	Description
no_color_video	ncv	NC	Video attributes that can't be used with colors
num_labels	nlab	Nl	Number of labels on screen (start at 1)
number_of_pins	npins	Yh	Number of pins in print-head
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
print_rate	cps	Ym	Print rate in characters per second
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	scesa	S8	Alternate escape for scancode emulation (default is for vt100)
back_tab	cbt	bt	Back tab

Variable	Cap-name	Termcap Code	Description
bell	bel	bl	Audible signal (bell)
bit_image_carriage_return	bicr	Yv	Move to beginning of same row (use tparm)
bit_image_newline	binel	Zz	Move to next row of the bit image (use tparm)
bit_image_repeat	birep	Zy	Repeat bit-image cell #1 #2 times (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

Variable	Cap-name	Termcap Code	Description
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
create_window	cwin	CW	Define win #1 to go from #2,#3 to #4,#5
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

Variable	Cap-name	Termcap Code	Description
device_type	devt	dv	Indicate language/codeset support
dial_phone	dial	DI	Dial phone number #1
dis_status_line	dsl	ds	Disable status line
display_clock	dclk	DK	Display time-of-day clock
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 mode
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

Variable	Cap-name	Termcap Code	Description
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
			mode
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

Variable	Cap-name	Termcap Code	Description
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
fixed_pause	pause	PA	Pause for 2-3 seconds
flash_hook	hook	fh	Flash the switch hook
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
get_mouse	getm	Gm	Curses should get button events
goto_window	wingo	WG	Go to window #1
hangup	hup	HU	Hang-up phone
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

Variable	Cap-name	Termcap Code	Description
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(3CURSES)).

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

Variable	Cap-name	Termcap Code	Description
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

Variable	Cap-name	Termcap Code	Description
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	kf4	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

Variable	Cap-name	Termcap Code	Description
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

Variable	Cap-name	Termcap Code	Description
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	kf40	FU	KEY_F(40), sent by function key fB0
key_fB1	kf41	FV	KEY_F(41), sent by function key fB1
key_fB2	kf42	FW	KEY_F(42), sent by function key fB2
key_fB3	kf43	FX	KEY_F(43), sent by function key fB3
key_fB4	kf44	FY	KEY_F(44), sent by function key fB4
key_fB5	kf45	FZ	KEY_F(45), sent by function key fB5
key_fB6	kf46	Fa	KEY_F(46), sent by function key fB6
key_fB7	kf47	Fb	KEY_F(47), sent by function key fB7
key_fB8	kf48	Fc	KEY_F(48), sent by function key fB8
key_fB9	kf49	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

Variable	Cap-name	Termcap Code	Description
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

Variable	Cap-name	Termcap Code	Description
			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_mouse	kmous	Km	0631, Mouse event has occured
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

Variable	Cap-name	Termcap Code	Description
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

Variable	Cap-name	Termcap Code	Description
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

Variable	Cap-name	Termcap Code	Description
			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

Variable	Cap-name	Termcap Code	Description
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_format	fln	Lf	Label format
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
mouse_info	minfo	Mi	Mouse status information
newline	nel	nw	Newline (behaves like `cr` followed by `lf`)

Variable	Cap-name	Termcap Code	Description
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

Variable	Cap-name	Termcap Code	Description
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
pulse	pulse	PU	Select pulse dialing
quick_dial	qdial	QD	Dial phone number #1, without
			progress detection
remove_clock	rmclk	RC	Remove time-of-day clock
repeat_char	rep	rp	Repeat char #1 #2 times
req_for_input	rfi	RF	Send next input char (for ptys)
req_mouse_pos	reqmp	RQ	Request mouse position report
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

Variable	Cap-name	Termcap Code	Description
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
			attributes #1-#6
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_clock	sclk	SC	Set time-of-day clock
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

Variable	Cap-name	Termcap Code	Description
set_page_length	slines	YZ	Set page length to #1 lines (use tparm)
			of an inch
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
tone	tone	TO	Select touch tone dialing
user0	u0	u0	User string 0

Variable	Cap-name	Termcap Code	Description
user1	u1	u1	User string 1
user2	u2	u2	User string 2
user3	u3	u3	User string 3
user4	u4	u4	User string 4
user5	u5	u5	User string 5
user6	u6	u6	User string 6
user7	u7	u7	User string 7
user8	u8	u8	User string 8
user9	u9	u9	User string 9
underline_char	uc	uc	Underscore one char and move past it
up_half_line	hu	hu	Half-line up (reverse 1/2 linefeed)
wait_tone	wait	WA	Wait for dial tone
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&T610;80column;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[0m^O, is3=\E(B\E)0, kLFT=\E[\s@, kRIT=\E[\sA,
   kbs=^H, 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, kf4=\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=^O, 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%t^N%e^O%;,
   sgr0=\E[m^O, smacs=^N, 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 integer2 op pop integer1)
`%& %| %^`	bit operations: push(pop integer2 op pop integer1)
`%= %> %<`	logical operations: push(pop integer2 op pop integer1)

`%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%;

sequence	when to output	terminfo translation
;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.

	vt100+
glyph name	**character**
arrow pointing right	+
arrow pointing left	,
arrow pointing down	.
solid square block	0
lantern symbol	I

	vt100+
glyph name	**character**
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,

	vt100+	**new tty**
glyph name	**char**	**char**
upper left corner	l	R
lower left corner	m	F
upper right corner	k	T

glyph name	vt100+ char	new tty char
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|Teletype4424 in display function group ii,
rev@, sgr@, smul@, use=att4424,
```

defines an AT&T4424 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):
sp
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 B_i 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:	
$bold spinv ’$	$bold spinv$

(continued)

(Continuation)

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), `infocmp`(1M), `tic`(1M), `printf`(3C), `curses`(3CURSES), `curses`(3XCURSES)

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 `timezone` file does not set the timezone environment variable TZ. See TIMEZONE(4) for information to set the TZ environment variable.

EXAMPLES

EXAMPLE 1 A sample display of `timezone` command.

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

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

tnf_kernel_probes – TNF kernel probes

DESCRIPTION

The set of probes (trace instrumentation points) available in the standard kernel. The probes log trace data to a kernel trace buffer in Trace Normal Form (TNF). Kernel probes are controlled by `prex`(1). A snapshot of the kernel trace buffer can be made using `tnfxtract`(1) and examined using `tnfdump`(1).

Each probe has a *name* and is associated with a set of symbolic *keys*, or *categories*. These are used to select and control probes from `prex`(1). A probe that is enabled for tracing generates a TNF record, called an *event record*. An event record contains two common members and may contain other probe-specific data members.

Common Members

```
tnf_probe_event    tag
tnf_time_delta     time_delta
```

tag — Encodes TNF references to two other records:

- *tag* — Describes the layout of the event record.
- *schedule* — Identifies the writing thread and also contains a 64-bit base time in nanoseconds.

time_delta — A 32-bit time offset from the base time; the sum of the two times is the actual time of the event.

Threads

`thread_create`

```
tnf_kthread_id    tid
tnf_pid           pid
tnf_symbol        start_pc
```

Thread creation event.

tid — The thread identifier for the new thread.

pid — The process identifier for the new thread.

start_pc — The kernel address of its start routine.

`thread_state`

```
tnf_kthread_id    tid
tnf_microstate    state
```

Thread microstate transition events.

tid — Optional; if it is absent, the event is for the writing thread, otherwise the event is for the specified thread.

state — Indicates the thread state:

- Running in user mode.
- Running in system mode.
- Asleep waiting for a user-mode lock.

- Asleep on a kernel object.
- Runnable (waiting for a cpu).
- Stopped.

The values of this member are defined in `<sys/msacct.h>`. Note that to reduce trace output, transitions between the *system* and *user* microstates that are induced by system calls are not traced. This information is implicit in the system call entry and exit events.

thread_exit

Thread termination event for writing thread. This probe has no data members other than the common members.

Scheduling

thread_queue

```
tnf_kthread_id    tid
tnf_cpuid         cpuid
tnf_long          priority
tnf_ulong         queue_length
```

Thread scheduling events. These are triggered when a runnable thread is placed on a dispatch queue.

cpuid Specifies the cpu to which the queue is attached.

priority The (global) dispatch priority of the thread.

queue_length The current length of the cpu's dispatch queue.

Blocking

`thread_block`

```
tnf_opaque     reason
tnf_symbols    stack
```

Thread blockage event. This probe captures a partial stack backtrace when the current thread blocks.

reason The address of the object on which the thread is blocking.

symbols References a TNF array of kernel addresses representing the PCs on the stack at the time the thread blocks.

System Calls

`syscall_start`

```
tnf_sysnum    sysnum
```

System call entry event.

sysnum The system call number. The writing thread implicitly enters the *system* microstate with this event.

`syscall_end`

```
tnf_long     rval1
tnf_long     rval2
tnf_long     errno
```

System call exit event.

rval1 and *rval2* — The two return values of the system call

errno — The error return.

The writing thread implicitly enters the *user* microstate with this event.

Page Faults

`address_fault`

```
tnf_opaque       address
tnf_fault_type   fault_type
tnf_seg_access   access
```

Address-space fault event.

address — Gives the faulting virtual address.

fault_type — Gives the fault type: invalid page, protection fault, software requested locking or unlocking.

access — Gives the desired access protection: read, write, execute or create. The values for these two members are defined in `<vm/seg_enum.h>`.

`major_fault`

```
tnf_opaque   vnode
tnf_offset   offset
```

Major page fault event. The faulting page is mapped to the file given by the *vnode* member, at the given *offset* into the file. (The faulting virtual address is in the most recent `address_fault` event for the writing thread.)

`anon_private`

```
tnf_opaque   address
```

Copy-on-write page fault event.

address — The virtual address at which the new page is mapped.

`anon_zero`

```
tnf_opaque   address
```

Zero-fill page fault event.

address — The virtual address at which the new page is mapped.

`page_unmap`

```
tnf_opaque   vnode
tnf_offset   offset
```

Page unmapping event. This probe marks the unmapping of a file system page from the system.

vnode and *offset* — Identifies the file and offset of the page being unmapped.

Pageins and Pageouts

pagein

```
tnf_opaque    vnode
tnf_offset    offset
tnf_size      size
```

Pagein start event. This event signals the initiation of pagein I/O.

vnode and *offset* — Identifyies the file and offset to be paged in.

size — Specifies the number of bytes to be paged in.

pageout

```
tnf_opaque    vnode
tnf_ulong     pages_pageout
tnf_ulong     pages_freed
tnf_ulong     pages_reclaimed
```

Pageout completion event. This event signals the completion of pageout I/O.

vnode — Identifies the file of the pageout request.

pages_pageout — The number of pages written out.

pages_freed — The number of pages freed after being written out.

pages_reclaimed — The number of pages reclaimed after being written out.

Page Daemon (Page Stealer)

pageout_scan_start

```
tnf_ulong    pages_free
tnf_ulong    pages_needed
```

Page daemon scan start event. This event signals the beginning of one iteration of the page daemon.

pages_free — The number of free pages in the system.

pages_needed — The number of pages desired free.

pageout_scan_end

```
tnf_ulong    pages_free
tnf_ulong    pages_scanned
```

Page daemon scan end event. This event signals the end of one iteration of the page daemon.

pages_free — The number of free pages in the system.

pages_scanned The number of pages examined by the page daemon. (Potentially more pages will be freed when any queued pageout requests complete.)

Swapper

`swapout_process`

```
tnf_pid        pid
tnf_ulong      page_count
```

Address space swapout event. This event marks the swapping out of a process address space.

pid Identifies the process.

page_count Reports the number of pages either freed or queued for pageout.

`swapout_lwp`

```
tnf_pid           pid
tnf_lwpid         lwpid
tnf_kthread_id    tid
tnf_ulong         page_count
```

Light-weight process swapout event. This event marks the swapping out of an LWP and its stack.

pid The LWP's process identifier

lwpid The LWP identifier

tid member The LWP's kernel thread identifier.

page_count The number of pages swapped out.

`swapin_lwp`

```
tnf_pid           pid
tnf_lwpid         lwpid
tnf_kthread_id    tid
tnf_ulong         page_count
```

Light-weight process swapin event. This event marks the swapping in of an LWP and its stack.

pid The LWP's process identifier.

lwpid The LWP identifier.

tid The LWP's kernel thread identifier.

page_count The number of pages swapped in.

Local I/O

`strategy`

```
tnf_device        device
tnf_diskaddr      block
```

```
tnf_size         size
tnf_opaque       buf
tnf_bioflags     flags
```

Block I/O strategy event. This event marks a call to the `strategy`(9E) function of a block device driver.

device Contains the major and minor numbers of the device.

block The logical block number to be accessed on the device.

size The size of the I/O request.

buf The kernel address of the `buf`(9S) structure associated with the transfer.

flags The `buf`(9S) flags associated with the transfer.

`biodone`

```
tnf_device      device
tnf_diskaddr    block
tnf_opaque      buf
```

Buffered I/O completion event. This event marks calls to the `biodone`(9F) function.

device Contains the major and minor numbers of the device.

block The logical block number accessed on the device.

buf The kernel address of the `buf`(9S) structure associated with the transfer.

`physio_start`

```
tnf_device      device
tnf_offset      offset
tnf_size        size
tnf_bioflags    rw
```

Raw I/O start event. This event marks entry into the `physio`(9F) fufnction which performs unbuffered I/O.

device Contains the major and minor numbers of the device of the transfer.

offset The logical offset on the device for the transfer.

size The number of bytes to be transferred.

rw The direction of the transfer: read or write (see `buf`(9S)).

`physio_end`

```
tnf_device     device
```

Raw I/O end event. This event marks exit from the `physio`(9F) fufnction.

device The major and minor numbers of the device of the transfer.

USAGE

Use the `prex` utility to control kernel probes. The standard `prex` commands to list and manipulate probes are available to you, along with commands to set up and manage kernel tracing.

Kernel probes write trace records into a kernel trace buffer. You must copy the buffer into a TNF file for post-processing; use the `tnfxtract` utility for this.

You use the `tnfdump` utility to examine a kernel trace file. This is exactly the same as examining a user-level trace file.

The steps you typically follow to take a kernel trace are:

1. Become superuser (`su`).
2. Allocate a kernel trace buffer of the desired size (`prex`).
3. Select the probes you want to trace and enable (`prex`).
4. Turn kernel tracing on (`prex`).
5. Run your application.
6. Turn kernel tracing off (`prex`).
7. Extract the kernel trace buffer (`tnfxtract`).
8. Disable all probes (`prex`).
9. Deallocate the kernel trace buffer (`prex`).
10. Examine the trace file (`tnfdump`).

A convenient way to follow these steps is to use two shell windows; run an interactive `prex` session in one, and run your application and `tnfxtract` in the other.

SEE ALSO

`prex`(1), `tnfdump`(1), `tnfxtract`(1), `libtnfctl`(3TNF), `TNF_PROBE`(3TNF), `tracing`(3TNF), `strategy`(9E), `biodone`(9F), `physio`(9F), `buf`(9S)

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

EXAMPLE 1 A sample from a configuration file.

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)

System Administration Guide, Volume 1 System Interface 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`(7I) 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), `termio`(7I)

System Administration Guide, Volume 1

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

EXAMPLE 1 A sample display of `/etc/ttysrch` command.

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;
```

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 %[^\ ]\ "
```

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`(3C) to write an entry to `/etc/dumpdates` ; `DUMPINFMT` is the format to use when using `scanf`(3C) to read an entry from `/etc/dumpdates` .

ATTRIBUTES

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

ATTRIBUTE TYPE	ATTRIBUTE VALUE
Stability Level	Unstable

SEE ALSO

`ufsdump`(1M) , `ufsrestore`(1M) , `ctime`(3C) , `printf`(3C) , `scanf`(3C) , `attributes`(5) , `types`(3HEAD)

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 the Network Information Service (NIS) databases. Databases can only be updated in a secure network, that is, 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 an NIS database named `passwd.byname` that can be updated, there should be a `make` target named `passwd.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 4 and 6):

1. Network name of client wishing to make the update (a string).
2. Kind of update (an integer).
3. Number of bytes in key (an integer).
4. Actual bytes of key.
5. Number of bytes in data (an integer).
6. Actual bytes of data.

After receiving this information through standard input, the command to update the particular database determines whether the user is allowed to make the change. If not, it exits with the status `YPERR_ACCESS.` If the user is allowed to make the change, the command makes the change and exits with a status of zero. If there are any errors that may prevent the `updaters` 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`	The makefile used for updating the NIS databases.

SEE ALSO

`make`(1S), `rpc.ypupdated`(1M), `publickey`(4)

NOTES

The Network Information Service (NIS) was formerly known as Sun Yellow Pages (YP). The functionality of the two remains the same; only the name has changed. The name Yellow Pages is a registered trademark in the United Kingdom of British Telecommunications plc, and may not be used without permission.

NAME

user_attr – extended user attributes database

SYNOPSIS

/etc/user_attr

DESCRIPTION

/etc/user_attr is a local source of extended attributes associated with users and roles. user_attr can be used with other user attribute sources, including the user_attr NIS map and NIS+ table. Programs use the getuserattr(3SECDB) routines to gain access to this information.

The search order for multiple user_attr sources is specified in the /etc/nsswitch.conf file, as described in the nsswitch.conf(4) man page. The search order follows that for passwd(4).

Each entry in the user_attr databases consists of a single line with five fields separated by colons (:). Line continuations using the backslash (\) character are permitted. Each entry has the form:

user:*qualifier*:*res1*:*res2*:*attr*

user — The name of the user as specified in the passwd(4) database.

qualifier — Reserved for future use.

res1 — Reserved for future use.

res2 — Reserved for future use.

attr — An optional list of semicolon-separated (;) key-value pairs that describe the security attributes to apply to the object upon execution. Zero or more keys may be specified. There are four valid keys: auths, profiles, roles, type.

- auths — Specifies a comma-separated list of authorization names chosen from those names defined in the auth_attr(4) database. Authorization names may be specified using the asterisk (*) character as a wildcard. For example, solaris.printer.* means all of Sun's printer authorizations.

- profiles — Contains an ordered, comma-separated list of profile names chosen from prof_attr(4). Profiles are enforced by the profile shells, pfcsh, pfksh, and pfsh. (See pfsh(1).) If no profiles are assigned, the profile shells do not allow the user to execute any commands.

roles — Can be assigned a comma-separated list of role names from the set of user accounts in this database whose `type` field indicates the account is a role. If the `roles` key value is not specified, the user is not permitted to assume any role.

type — Can be assigned one of these strings: `normal`, indicating that this account is for a normal user, one who logs in; or `role`, indicating that this account is for a role. Roles can only be assumed by a normal user after the user has logged in.

EXAMPLES

EXAMPLE 1 Assigning a profile to root

The following example entry assigns to root the `All` profile, which allows root to use all commands in the system, and also assigns two authorizations:

```
root::::auths=solaris.*,solaris.grant;profiles=All;type=normal
```

The `solaris.*` wildcard authorization shown above gives root all the `solaris` authorizations; and the `solaris.grant` authorization gives root the right to grant to others any `solaris` authorizations that root has. The combination of authorizations enables root to grant to others all the `solaris` authorizations. See `auth_attr`(4) for more about authorizations.

FILES

`/etc/nsswitch.conf`

`/etc/user_attr`

NOTES

When deciding which authorization source to use (see DESCRIPTION), keep in mind that NIS+ provides stronger authentication than NIS.

The root user is usually defined in local databases for a number of reasons, including the fact that root needs to be able to log in and do system maintenance in single-user mode, before the network name service databases are available. For this reason, an entry should exist for root in the local `user_attr` file, and the precedence shown in the example `nsswitch.conf`(4) file entry under EXAMPLES is highly recommended.

Because the list of legal keys is likely to expand, any code that parses this database must be written to ignore unknown key-value pairs without error. When any new keywords are created, the names should be prefixed with a unique string, such as the company's stock symbol, to avoid potential naming conflicts.

In the `attr` field, escape the following symbols with a backslash (\) if you use them in any value: colon (`:`), semicolon (`;`), carriage return (`\n`), equals (`=`), or backslash (\).

SEE ALSO

`auths`(1), `pfcsh`(1), `pfksh`(1), `pfsh`(1), `profiles`(1), `roles`(1), `getuserattr`(3SECDB), `auth_attr`(4), `exec_attr`(4), `nsswitch.conf`(4), `passwd`(4), `prof_attr`(4)

NAME

utmp, wtmp – utmp and wtmp database entry formats

SYNOPSIS

`#include <utmp.h>/var/adm/utmp/var/adm/wtmp`

DESCRIPTION

The `utmp` and `wtmp` database files are obsolete and are no longer present on the system. They have been superseded by the extended database contained in the `utmpx` and `wtmpx` database files. See `utmpx`(4) .

It is possible for `/var/adm/utmp` to reappear on the system. This would most likely occur if a third party application that still uses `utmp` recreates the file if it finds it missing. This file should not be allowed to remain on the system. The user should investigate to determine which application is recreating this file.

SEE ALSO

`utmpx`(4)

NAME

utmpx, wtmpx – utmpx and wtmpx database entry formats

SYNOPSIS

`#include <utmpx.h>/var/adm/utmpx/var/adm/wtmpx`

DESCRIPTION

The `utmpx` and `wtmpx` files are extended database files that have superseded the obsolete `utmp` and `wtmp` database files.

The `utmpx` database contains user access and accounting information for commands such as `who`(1) , `write`(1) , and `login`(1) . The `wtmpx` database contains the history of user access and accounting information for the `utmpx` database.

USAGE

Applications should not access these databases directly, but should use the functions described on the `getutxent`(3C) manual page to interact with the `utmpx` and `wtmpx` databases to ensure that they are maintained consistently.

FILES

`/var/adm/utmpx`	user access and adminstration information
`/var/adm/wtmpx`	history of user access and adminstrative information

SEE ALSO

`wait`(2) , `getutxent`(3C) , `wstat`(3XFN)

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       device       mount      FS      fsck    mount      mount
to mount     to fsck      point      type    pass    at boot    options
```

The fields in the table are space-separated and show the resource name (*device to mount*), the raw device to `fsck` (*device to fsck*), the default mount directory (*mount point*), the name of the file system type (*FS type*), the number used by `fsck` to decide whether to check the file system automatically (*fsck pass*), whether the file system should be mounted automatically by `mountall` (*mount at boot*), and the file system mount options (*mount options*). (See respective mount file system man page below in SEE ALSO for *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), `getvfsent`(3C)

System Administration Guide, Volume 1

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

EXAMPLE 1 A sample `vold.conf` file.

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/cdroms`*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

`sh`(1), `volcancel`(1), `volcheck`(1), `volmissing`(1), `rmmount`(1M), `vold`(1M), `rmmount.conf`(4), `volfs`(7FS)

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. If you follow the conventions specified below, Volume Management figures out both device names if only one of them is specified. For example, if you specify the block device, it figures out the pathname to the character device; if you specify the pathname to the character device, it figures out the block device.

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 the character name given the block name.

NAME

warn.conf – Kerberos warning configuration file

SYNOPSIS

/etc/krb5/warn.conf

DESCRIPTION

The `warn.conf` file contains configuration information specifying how users will be warned by the `ktkt_warnd` daemon about ticket expiration on a Kerberos client. Each Kerberos client host must have a `warn.conf` file in order for users on that host to get Kerberos warnings from the client. Entries in the `warn.conf` file must have the following format:

principal *syslog* | *terminal* | `mail` `time` [*email_address*]

principal — The principal name to be warned. The '*' wildcard can be used to specify groups of principals.

syslog — Sends the warnings to the system's syslog. Depending on the `/etc/syslog.conf` file, syslog entries are written to the `/var/adm/messages` file and/or displayed on the terminal.

terminal — Sends the warnings to display on the terminal.

`mail` — Sends the warnings as email to the address specified by *email_address*.

`time` — Specifies how much time before the TGT expires when a warning should be sent. The default time value is seconds, but you can specify `h` (hours) and `m` (minutes) after the number to specify other time values.

email_address — Specifies the email address at which to send the warnings. This field must be specified only with the `mail` field.

EXAMPLES

EXAMPLE 1 Specifying warnings

The following `warn.conf` entry specifies that warnings will be sent to the syslog 5 minutes before the expiration of the TGT for all principals, in the form: "`jdb@ACME.COM: your kerberos credentials expire in 5 minutes`".

```
* syslog 5m
```

FILES

`/usr/lib/krb5/ktkt_warnd` — Kerberos warning daemon

SEE ALSO

`ktkt_warnd`(1M), SEAM(5)

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 (in ASCII form) 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`(3NSL)). 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 subdirectory of `/var/yp` containing a set of NIS maps on each NIS server.

Standard nicknames are defined in the file `/var/yp/nicknames`. These names can be used in place of the full map name in the `ypmatch` and `ypcat` commands. The command `ypwhich` `–m` can be used to display the full set of nicknames. Each line of the nickname file contains two fields separated by white space. The first field is the nickname and the second field is the name of the map that it expands to. The nickname cannot contain a ".".

FILES

`/var/yp/nicknames` nicknames file

SEE ALSO

`nis+`(1), `nisaddent`(1M), `nissetup`(1M), `rpc.nisd`(1M), `ypbind`(1M), `ypinit`(1M), `dbm`(3UCB), `secure_rpc`(3NSL), `ypclnt`(3NSL)

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

NAME

zoneinfo – timezone information

DESCRIPTION

For notes regarding the zoneinfo timezones, see `/usr/share/lib/zoneinfo/src/README`.

Index

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T