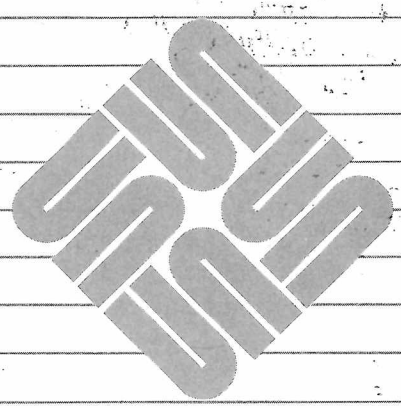
sun® microsystems

SunOS Reference Manual



Part Number: 800-3827-10
Revision A of 27 March, 1990

The Sun logo, Sun Microsystems, Sun Workstation, NFS, and TOPS are registered trademarks of Sun Microsystems, Inc.

Sun, Sun-2, Sun-3, Sun-4, Sun386*i*, SPARCstation, SPARCserver, NeWS, NSE, OpenWindows, SPARC, SunInstall, SunLink, SunNet, SunOS, SunPro, and SunView are trademarks of Sun Microsystems, Inc.

UNIX is a registered trademark of AT&T; OPEN LOOK is a trademark of AT&T.

All other products or services mentioned in this document are identified by the trademarks or service marks of their respective companies or organizations, and Sun Microsystems, Inc. disclaims any responsibility for specifying which marks are owned by which companies or organizations.

This logo is a trademark of the X/Open Company Limited in the UK and other countries, and its use is licensed to Sun Microsystems, Inc. The use of this logo certifies SunOS 4.1 conformance with X/Open Portability Guide Issue 2 (XPG 2).

Copyright © 1987, 1988, 1989, 1990 Sun Microsystems, Inc. – Printed in U.S.A.

All rights reserved. No part of this work covered by copyright hereon may be reproduced in any form or by any means – graphic, electronic, or mechanical – including photocopying, recording, taping, or storage in an information retrieval system, without the prior written permission of the copyright owner.

Restricted rights legend: use, duplication, or disclosure by the U.S. government is subject to restrictions set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 52.227-7013 and in similar clauses in the FAR and NASA FAR Supplement.

The 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.

This product is protected by one or more of the following U.S. patents: 4,777,485 4,688,190 4,527,232 4,745,407 4,679,014 4,435,792 4,719,569 4,550,368 in addition to foreign patents and applications pending.

This software and documentation is based in part on the Fourth Berkeley Software Distribution under license from the Regents of the University of California. We acknowledge the following individuals and institutions for their role in its development: The Regents of the University of California, the Electrical Engineering and Computer Sciences Department at the Berkeley Campus of the University of California, and Other Contributors.

NAME

intro – file formats used or read by various programs

DESCRIPTION

This section describes formats of files used by various programs.

A 5V section number means one or more of the following:

- The man page documents System V formats only.
- The man page documents default SunOS formats, and System V formats as they differ from the default formats. These System V differences are presented under **SYSTEM V** section headers.
- The man page documents formats compliant with *IEEE Std 1003.1-1988* (POSIX.1).

LIST OF FILE FORMATS

Name	Appears on Page	Description
acct	**acct**(5)	execution accounting file
addresses	**aliases**(5)	addresses and aliases for sendmail
aliases	**aliases**(5)	addresses and aliases for sendmail
a.out	**a.out**(5)	assembler and link editor output format
ar	**ar**(5)	archive (library) file format
audit_control	**audit_control**(5)	control information for system audit daemon
audit_data	**audit_data**(5)	current information on audit daemon
audit.log	**audit.log**(5)	the security audit trail file
auto.home	**auto.home**(5)	automount map for home directories
auto.vol	**auto.vol**(5)	automount map for volumes
bar	**bar**(5)	tape archive file format
boards.pc	**boards.pc**(5)	AT- and XT-compatible boards for DOS windows
bootparams	**bootparams**(5)	boot parameter data base
bootservers	**bootservers**(5)	NIS bootservers file
coff	**coff**(5)	common assembler and link editor output
core	**core**(5)	format of memory image file
cpio	**cpio**(5)	format of cpio archive
crontab	**crontab**(5)	table of times to run periodic jobs
dir	**dir**(5)	format of directories
dump	**dump**(5)	incremental dump format
dumpdates	**dump**(5)	incremental dump format
environ	**environ**(5V)	user environment
ethers	**ethers**(5)	Ethernet address to hostname database or NIS domain
exports	**exports**(5)	directories to export to NFS clients
ext_ports	**ext_ports**(5)	external ports file for network printers, terminals, and modems
fbtab	**fbtab**(5)	framebuffer table
fcntl	**fcntl**(5)	file control options
forward	**aliases**(5)	addresses and aliases for sendmail
fs	**fs**(5)	format of a 4.2 (ufs) file system volume
fspec	**fspec**(5)	format specification in text files
fstab	**fstab**(5)	static filesystem mounting table, mounted filesystems table
ftpusers	**ftpusers**(5)	list of users prohibited by FTP
gettytab	**gettytab**(5)	terminal configuration data base
group	**group**(5)	group file
group.adjunct	**group.adjunct**(5)	group security data file
help	**help**(5)	help file format
help_viewer	**help_viewer**(5)	help viewer file format
hosts	**hosts**(5)	host name data base
hosts.equiv	**hosts.equiv**(5)	trusted hosts by system and by user

indent.pro	**indent.pro**(5)	default options for indent
inetd.conf	**inetd.conf**(5)	Internet servers database
inode	**fs**(5)	format of a 4.2 (ufs) file system volume
internat	**internat**(5)	key mapping table for internationalization
ipalloc.netrange	**ipalloc.netrange**(5)	range of addresses to allocate
keytables	**keytables**(5)	keyboard table descriptions for loadkeys and dumpkeys
lastlog	**utmp**(5V)	login records
link	**link**(5)	link editor interfaces
locale	**locale**(5)	locale database
magic	**magic**(5)	file command's magic number file
mtab	**fstab**(5)	static filesystem mounting table, mounted filesystems table
mtab	**mtab**(5)	mounted file system table
netgroup	**netgroup**(5)	list of network groups
netmasks	**netmasks**(5)	network mask data base
netrc	**netrc**(5)	file for ftp remote login data
networks	**networks**(5)	network name data base
orgrc	**orgrc**(5)	organizer configuration and initialization file
passwd	**passwd**(5)	password file
passwd.adjunct	**passwd.adjunct**(5)	user security data file
phones	**phones**(5)	remote host phone number data base
plot	**plot**(5)	graphics interface
pnp.sysnames	**pnp.sysnames**(5)	file used to allocate system names
policies	**policies**(5)	network administration policies
printcap	**printcap**(5)	printer capability data base
proto	**proto**(5)	prototype job file for at
protocols	**protocols**(5)	protocol name data base
publickey	**publickey**(5)	public key database
queuedefs	**queuedefs**(5)	queue description file for at, batch, and cron
rasterfile	**rasterfile**(5)	Sun's file format for raster images
remote	**remote**(5)	remote host description file
resolv.conf	**resolv.conf**(5)	configuration file for domain name system resolver
rfmaster	**rfmaster**(5)	Remote File Sharing name server master file
rgb	**rgb**(5)	available colors (by name) for coloredit
rhosts	**hosts.equiv**(5)	trusted hosts by system and by user
rootmenu	**rootmenu**(5)	root menu specification for SunView
rpc	**rpc**(5)	rpc program number data base
sccsfile	**sccsfile**(5)	format of an SCCS history file
services	**services**(5)	Internet services and aliases
setup.pc	**setup.pc**(5)	master configuration file for DOS
sm	**sm**(5)	in.statd directory and file structures
sm	**statmon**(5)	statd directories and file structures
sm.bak	**sm**(5)	in.statd directory and file structures
sm.bak	**statmon**(5)	statd directories and file structures
sm.state	**sm**(5)	in.statd directory and file structures
state	**statmon**(5)	statd directories and file structures
sunview	**sunview**(5)	initialization file for SunView
svdtab	**svdtab**(5)	SunView device table
syslog.conf	**syslog.conf**(5)	configuration file for syslogd system log daemon
systems	**systems**(5)	NIS systems file
tar	**tar**(5)	tape archive file format
termcap	**termcap**(5)	terminal capability data base
term	**term**(5)	terminal driving tables for nroff
term	**term**(5V)	format of compiled term file

terminfo	**terminfo**(5V)	terminal capability data base
toc	**toc**(5)	table of contents of optional clusters
translate	**translate**(5)	input and output files for system message translation
ttys	**ttytab**(5)	terminal initialization data
ttytab	**ttytab**(5)	terminal initialization data
types	**types**(5)	primitive system data types
tzfile	**tzfile**(5)	time zone information
ugid_alloc.range	**ugid_alloc.range**(5)	range of user IDs and group IDs to allocate
updaters	**updaters**(5)	configuration file for NIS updating
utmp	**utmp**(5V)	login records
uuencode	**uuencode**(5)	format of an encoded uuencode file
vfont	**vfont**(5)	font formats
vgrindefs	**vgrindefs**(5)	vgrind's language definition data base
wtmp	**utmp**(5V)	login records
xtab	**exports**(5)	directories to export to NFS clients
ypaliases	**ypaliases**(5)	NIS aliases for sendmail
ypfiles	**ypfiles**(5)	NIS database and directory structure
ypgroup	**ypgroup**(5)	NIS group file
yppasswd	**yppasswd**(5)	NIS password file
ypprintcap	**ypprintcap**(5)	NIS printer capability database

NAME

a.out – assembler and link editor output format

SYNOPSIS

```
#include <a.out.h>
#include <stab.h>
#include <nlist.h>
```

AVAILABILITY

Sun-2, Sun-3, and Sun-4 systems only. For Sun386i systems refer to **coff**(5).

DESCRIPTION

a.out is the output format of the assembler **as**(1) and the link editor **ld**(1). The link editor makes **a.out** executable files.

A file in **a.out** format consists of: a header, the program text, program data, text and data relocation information, a symbol table, and a string table (in that order). In the header, the sizes of each section are given in bytes. The last three sections may be absent if the program was loaded with the –s option of **ld** or if the symbols and relocation have been removed by **strip**(1).

The machine type in the header indicates the type of hardware on which the object code can be executed. Sun-2 code runs on Sun-3 systems, but not vice versa. Program files predating release 3.0 are recognized by a machine type of '**0**'. Sun-4 code may not be run on Sun-2 or Sun-3, nor vice versa.

Header

The header consists of a **exec** structure. The **exec** structure has the form:

```
struct exec {
        unsigned char   a_dynamic:1;    /* has a __DYNAMIC */
        unsigned char   a_toolversion:7; /* version of toolset used to create this file */
        unsigned char   a_machtype;     /* machine type */
        unsigned short  a_magic;        /* magic number */
        unsigned long   a_text;         /* size of text segment */
        unsigned long   a_data;         /* size of initialized data */
        unsigned long   a_bss;          /* size of uninitialized data */
        unsigned long   a_syms;         /* size of symbol table */
        unsigned long   a_entry;        /* entry point */
        unsigned long   a_trsize;       /* size of text relocation */
        unsigned long   a_drsize;       /* size of data relocation */
};
```

The members of the structure are:

a_dynamic 1 if the **a.out** file is dynamically linked or is a shared object, 0 otherwise.

a_toolversion The version number of the toolset (**as**, **ld**, etc.) used to create the file.

a_machtype One of the following:

0	pre-3.0 executable image
M_68010	executable image using only MC68010 instructions that can run on Sun-2 or Sun-3 systems.
M_68020	executable image using MC68020 instructions that can run only on Sun-3 systems.
M_SPARC	executable image using SPARC instructions that can run only on Sun-4 systems.

a_magic One of the following:

OMAGIC	An text executable image which is not to be write-protected, so the data segment is immediately contiguous with the text segment.

NMAGIC A write-protected text executable image. The data segment begins at the first segment boundary following the text segment, and the text segment is not writable by the program. When the image is started with **execve**(2V), the entire text and data segments will be read into memory.

ZMAGIC A page-aligned text executable image. the data segment begins at the first segment boundary following the text segment, and the text segment is not writable by the program. The text and data sizes are both multiples of the page size, and the pages of the file will be brought into the running image as needed, and not pre-loaded as with the other formats. This is the default format produced by **ld**(1).

The macro **N_BADMAG** takes an **exec** structure as an argument; it evaluates to 1 if the **a_magic** field of that structure is invalid, and evaluates to 0 if it is valid.

a_text The size of the text segment, in bytes.

a_data The size of the initialized portion of the data segment, in bytes.

a_bss The size of the "uninitialized" portion of the data segment, in bytes. This portion is actually initialized to zero. The zeroes are not stored in the **a.out** file; the data in this portion of the data segment is zeroed out when it is loaded.

a_syms The size of the symbol table, in bytes.

a_entry The virtual address of the entry point of the program; when the image is started with **execve**, the first instruction executed in the image is at this address.

a_trsize The size of the relocation information for the text segment.

a_drsize The size of the relocation information for the data segment.

The macros **N_TXTADDR**, **N_DATADDR**, and **N_BSSADDR** give the memory addresses at which the text, data, and bss segments, respectively, will be loaded.

In the ZMAGIC format, the size of the header is included in the size of the text section; in other formats, it is not.

When an **a.out** file is executed, three logical segments are set up: the text segment, the data segment (with uninitialized data, which starts off as all 0, following initialized data), and a stack. For the ZMAGIC format, the header is loaded with the text segment; for other formats it is not.

Program execution begins at the address given by the value of the **a_entry** field.

The stack starts at the highest possible location in the memory image, and grows downwards. The stack is automatically extended as required. The data segment is extended as requested by **brk**(2) or **sbrk**.

Text and Data Segments

The text segment begins at the start of the file for ZMAGIC format, or just after the header for the other formats. The **N_TXTOFF** macro returns this absolute file position when given an **exec** structure as argument. The data segment is contiguous with the text and immediately followed by the text relocation and then the data relocation information. The **N_DATOFF** macro returns the absolute file position of the beginning of the data segment when given an **exec** structure as argument.

Relocation

The relocation information appears after the text and data segments. The **N_TRELOFF** macro returns the absolute file position of the relocation information for the text segment, when given an **exec** structure as argument. The **N_DRELOFF** macro returns the absolute file position of the relocation information for the data segment, when given an **exec** structure as argument. There is no relocation information if **a_trsize+a_drsize==0**.

Relocation (Sun-2 and Sun-3 Systems)

If a byte in the text or data involves a reference to an undefined external symbol, as indicated by the relocation information, then the value stored in the file is an offset from the associated external symbol. When

the file is processed by the link editor and the external symbol becomes defined, the value of the symbol is added to the bytes in the file. If a byte involves a reference to a relative location, or relocatable segment, then the value stored in the file is an offset from the associated segment.

If relocation information is present, it amounts to eight bytes per relocatable datum as in the following structure:

```
struct reloc_info_68k {
    long        r_address;          /* address which is relocated */
unsigned int    r_symbolnum:24,     /* local symbol ordinal */
    r_pcrel:1,          /* was relocated pc relative already */
    r_length:2,         /* 0=byte, 1=word, 2=long */
    r_extern:1,         /* does not include value of sym referenced */
    r_baserel:1,        /* linkage table relative */
    r_jmptable:1,       /* pc-relative to jump table */
    r_relative:1,       /* relative relocation */
    :1;
};
```

If **r_extern** is 0, then **r_symbolnum** is actually an **n_type** for the relocation (for instance, **N_TEXT** meaning relative to segment text origin.)

Relocation (Sun-4 System)

If a byte in the text or data involves a reference to an undefined external symbol, as indicated by the relocation information, then the value stored in the file is ignored. Unlike the Sun-2 and Sun-3 system, the offset from the associated symbol is kept with the relocation record. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol is added to this offset, and the sum is inserted into the bytes in the text or data segment.

If relocation information is present, it amounts to twelve bytes per relocatable datum as in the following structure:

```
enum reloc_type
{
    RELOC_8,            RELOC_16,           RELOC_32,           /* simplest relocs */
    RELOC_DISP8,        RELOC_DISP16,       RELOC_DISP32,       /* Disp's (pc-rel) */
    RELOC_WDISP30,      RELOC_WDISP22,      /* SR word disp's */
    RELOC_HI22,         RELOC_22,           /* SR 22-bit relocs */
    RELOC_13,           RELOC_LO10,         /* SR 13&10-bit relocs */
    RELOC_SFA_BASE,     RELOC_SFA_OFF13,    /* SR S.F.A. relocs */
    RELOC_BASE10,       RELOC_BASE13,       RELOC_BASE22,       /* base_relative pic */
    RELOC_PC10,         RELOC_PC22,         /* special pc-rel pic*/
    RELOC_JMP_TBL,      /* jmp_tbl_rel in pic */
    RELOC_SEGOFF16,     /* ShLib offset-in-seg */
    RELOC_GLOB_DAT,     RELOC_JMP_SLOT,     RELOC_RELATIVE,     /* rtld relocs */
};

struct reloc_info_sparc   /* used when header.a_machtype == M_SPARC */
{
    unsigned long int   r_address;          /* relocation addr (offset in segment) */
    unsigned int        r_index     :24;    /* segment index or symbol index */
    unsigned int        r_extern    : 1;    /* if F, r_index==SEG#; if T, SYM idx */
    int                 : 2;                /* <unused> */
    enum reloc_type     r_type      : 5;    /* type of relocation to perform */
    long int            r_addend;           /* addend for relocation value */
};
```

If **r_extern** is 0, then **r_index** is actually a **n_type** for the relocation (for instance, N_TEXT meaning relative to segment text origin.)

Symbol Table

The N_SYMOFF macro returns the absolute file position of the symbol table when given an **exec** structure as argument. Within this symbol table, distinct symbols point to disjoint areas in the string table (even when two symbols have the same name). The string table immediately follows the symbol table; the N_STROFF macro returns the absolute file position of the string table when given an **exec** structure as argument. The first 4 bytes of the string table are not used for string storage, but rather contain the size of the string table. This size *includes* the 4 bytes; thus, the minimum string table size is 4. Layout information as given in the include file for the Sun system is shown below.

The layout of a symbol table entry and the principal flag values that distinguish symbol types are given in the include file as follows:

```
struct nlist {
        union {
                char    *n_name;        /* for use when in-memory */
                long    n_strx;         /* index into file string table */
        } n_un;
        unsigned char   n_type;         /* type flag, that is, N_TEXT etc; see below */
        char    n_other;
        short   n_desc;                 /* see <stab.h> */
        unsigned        n_value;        /* value of this symbol (or adb offset) */
};
#define n_hash  n_desc                  /* used internally by ld */
/*
* Simple values for n_type.
*/
#define N_UNDF  0x0                     /* undefined */
#define N_ABS   0x2                     /* absolute */
#define N_TEXT  0x4                     /* text */
#define N_DATA  0x6                     /* data */
#define N_BSS   0x8                     /* bss */
#define N_COMM  0x12                    /* common (internal to ld) */
#define N_FN    0x1f                    /* file name symbol */
#define N_EXT   01                      /* external bit, or'ed in */
#define N_TYPE  0x1e                    /* mask for all the type bits */

/*
* Other permanent symbol table entries have some of the N_STAB bits set.
* These are given in <stab.h>
*/
#define N_STAB  0xe0                    /* if any of these bits set, don't discard */
```

In the **a.out** file a symbol's **n_un.n_strx** field gives an index into the string table. A **n_strx** value of 0 indicates that no name is associated with a particular symbol table entry. The field **n_un.n_name** can be used to refer to the symbol name only if the program sets this up using **n_strx** and appropriate data from the string table. Because of the union in the **nlist** declaration, it is impossible in C to statically initialize such a structure. If this must be done (as when using **nlist**(3V)) include the file **<nlist.h>**, rather than **<a.out.h>**. This contains the declaration without the union.

If a symbol's type is undefined external, and the value field is non-zero, the symbol is interpreted by the loader **ld** as the name of a common region whose size is indicated by the value of the symbol.

SEE ALSO

adb(1), **as**(1), **cc**(1V), **dbx**(1), **ld**(1), **nm**(1), **strip**(1), **brk**(2), **nlist**(3V), **coff**(5)

NAME

acct – execution accounting file

SYNOPSIS

#include <sys/acct.h>

DESCRIPTION

The **acct**(2V) system call makes entries in an accounting file for each process that terminates. The accounting file is a sequence of entries whose layout, as defined by the include file is:

```
typedef u_short comp_t;

struct acct
{
        char    ac_flag;        /* Accounting flag */
        char    ac_stat;        /* Exit status */
        uid_t   ac_uid;                 /* Accounting user ID */
        gid_t   ac_gid;                 /* Accounting group ID */
        dev_t   ac_tty;                 /* control typewriter */
        time_t  ac_btime;               /* Beginning time */
        comp_t  ac_utime;               /* Accounting user time */
        comp_t  ac_stime;               /* Accounting system time */
        comp_t  ac_etime;               /* Accounting elapsed time */
        comp_t  ac_mem;                         /* average memory usage */
        comp_t  ac_io;                  /* chars transferred */
        comp_t  ac_rw;                  /* blocks read or written */
        char    ac_comm[8];             /* Accounting command name */
};
```

The type **comp_t** is a 3 bits base 8 exponent, 13 bit fraction "floating point" number. If the process does an **execve**(2V), the first 8 characters of the filename appear in **ac_comm**. **ac_flag** contains bits indicating whether **execve**(2V) was ever accomplished, and whether the process ever had super-user privileges.

SEE ALSO

acct(2V), **execve**(2V), **sa**(8)

NAME

aliases, addresses, forward – addresses and aliases for sendmail

SYNOPSIS

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

DESCRIPTION

These files contain mail addresses or aliases, recognized by **sendmail**(8), for the local host:

/etc/passwd	Mail addresses (usernames) of local users.
/etc/aliases	Aliases for the local host, in ASCII format. This file can be edited to add, update, or delete local mail aliases.
/etc/aliases.{dir,pag}	The aliasing information from **/etc/aliases**, in binary, **dbm**(3X) format for use by **sendmail**(8). The program **newaliases**(8), which is invoked automatically by **sendmail**(8), maintains these files.
~/.forward	Addresses to which a user's mail is forwarded (see **Automatic Forwarding**, below).

In addition, the Network Information Service (NIS) aliases map *mail.aliases* contains addresses and aliases available for use across the network.

Addresses

As distributed, **sendmail**(8) 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**(8) pipes the message to the standard input of the *command* the bar precedes.

TCP/IP-*standard Addresses*

username@*domain*

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

.COM	Commercial organizations.
.EDU	Educational organizations.
.GOV	Government organizations.
.MIL	Military organizations.

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

js@jsmachine.Podunk-U.EDU

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

uucp(1C) *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** configuration file. See the **sendmail**(8), and *System and Network Administration* for details. Standard addresses are recommended.

Aliases

Local Aliases

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

aliasname: *address*[, *address*]

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

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

Special Aliases

An alias of the form:

owner–*aliasname*: *address*

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

An alias of the form:

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

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

NIS Domain Aliases

Normally, 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. Thus, the **/etc/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 client could just mail to **jsmith** and not have to remember the machine and username for John Smith. If an NIS alias does not resolve to an address with a specific host, then the name of the 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 an NIS client to **root@podunk-u** if the name of the 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** 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**(1) 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**(1) program.

FILES

/etc/passwd
/etc/aliases
~/.forward

SEE ALSO

uucp(1C), **vacation**(1), **dbm**(3X), **newaliases**(8), **sendmail**(8)

System and Network Administration

BUGS

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

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.

NAME

ar – archive (library) file format

SYNOPSIS

#include <ar.h>

DESCRIPTION

The archive command **ar** combines several files into one. Archives are used mainly as libraries to be searched by the link-editor **ld**(1).

A file produced by **ar** has a magic string at the start, followed by the constituent files, each preceded by a file header. The magic number and header layout as described in the include file are:

```
#define ARMAG   "!<arch>\n"
#define SARMAG 8

#define ARFMAG "`\n"

struct ar_hdr {
        char    ar_name[16];
        char    ar_date[12];
        char    ar_uid[6];
        char    ar_gid[6];
        char    ar_mode[8];
        char    ar_size[10];
        char    ar_fmag[2];
};
```

The name is a blank-padded string. The **ar_fmag** field contains ARFMAG to help verify the presence of a header. The other fields are left-adjusted, blank-padded numbers. They are decimal except for **ar_mode**, which is octal. The date is the modification date of the file at the time of its insertion into the archive.

Each file begins on a even (0 mod 2) boundary; a NEWLINE is inserted between files if necessary. Nevertheless the size given reflects the actual size of the file exclusive of padding.

There is no provision for empty areas in an archive file.

The encoding of the header is portable across machines. If an archive contains printable files, the archive itself is printable.

Sun386i DESCRIPTION

The file produced by *ar* on Sun386i systems is identical to that described above with the following changes:

Each archive containing COFF files [see **coff**(5)] includes an archive symbol table. This symbol table is used by the link editor **ld** to determine which archive members must be loaded during the link edit process. The archive symbol table (if it exists) is always the first file in the archive (but is never listed) and is automatically created and/or updated by **ar**.

The *ar_name* field of the ar_hdr structure described above is blank-padded and slash (/) terminated. Common format archives can be moved from system to system as long as the portable archive command **ar** is used. Conversion tools such as **convert** exist to aid in the transportation of non-common format archives to this format.

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

If the archive symbol table exists, the first file in the archive has a zero length name (i.e., **ar_name[0]** == '/'). The contents of this file are as follows:

- The number of symbols. Length: 4 bytes.
- The array of offsets into the archive file. Length: 4 bytes * "the number of symbols".

- The name string table. Length: *ar_size* – (4 bytes * ("the number of symbols" + 1)).

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

SEE ALSO

ar(1V), **ld**(1), **nm**(1)

Sun386i WARNINGS

strip(1) will remove all archive symbol entries from the header. The archive symbol entries must be restored via the ts option of the *ar*(1V) command before the archive can be used with the link editor *ld*(1).

BUGS

Filenames lose trailing blanks. Most software dealing with archives takes even an included blank as a name terminator.

NAME

audit.log – the security audit trail file

SYNOPSIS

```
#include <sys/label.h>
#include <sys/audit.h>
#include <sys/user.h>
```

DESCRIPTION

The **audit.log** file begins with a header record consisting of an **audit_header** structure followed by the previous audit file name. When the audit daemon is started (usually only at boot time), the previous audit file name is NULL.

```
struct audit_header {
        int     ah_magic;       /* magic number */
        time_t  ah_time;        /* the time */
        short   ah_namelen;     /* length of file name */
};
typedef struct audit_header audit_header_t;
```

The file may end with a trailer record consisting of an **audit_trailer** structure followed by the name of the next audit file.

```
struct audit_trailer {
        short   at_record_size;         /* size of this */
        short   at_record_type;         /* its type, a trailer */
        time_t  at_time;                /* the time */
        short   at_namelen;             /* length of file name */
};
typedef struct audit_trailer audit_trailer_t;
```

The **audit.log** file contains audit records in their raw form. The records are of varying size depending on the record type. Each record has a header which is an **audit_record** structure.

```
struct audit_record {
        short           au_record_size;         /* size of this */
        short           au_record_type;         /* its type */
        time_t          au_time;                /* the time */
        short           au_uid;                 /* real uid */
        short           au_auid;                /* audit uid */
        short           au_euid;                /* effective */
        short           au_gid;                 /* real group */
        short           au_pid;                 /* effective */
        int             au_errno;               /* error code */
        int             au_return;              /* a return value */
        blabel_t        au_label;               /* also ... */
        short           au_param_count;                 /* # of parameters */
};
typedef struct audit_record audit_record_t;
```

Immediately following the header is a set of two byte integers, the number of which exist for a given record is contained in the **au_param_count** field. These numbers are the lengths of the additional data items. The additional data items follow the list of lengths, the first length describing the first data item. Interpretation of this data is left to the program accessing it.

SEE ALSO

audit(2), **audit**(8)

Security Features Guide

NAME

audit_control – control information for system audit daemon

SYNOPSIS

/etc/security/audit/audit_control

DESCRIPTION

The **audit_control** file contains audit control information read by **auditd**(8). 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*

where *directory-name* is the name of a directory in which to create audit files, with the form:

/etc/security/audit/*server***/***machine*

where *server* is the name of an audit file system on the machine where this audit directory resides, and *machine* is the name of the local machine, since audit files belonging to different machines are, by convention, stored in separate subdirectories of a single audit directory. The naming convention normally has *server* be the name of 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.

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**(8) invokes the shell script **/etc/security/audit/audit_warn**. 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 **/etc/security/passwd.adjunct** 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 following table lists the audit classes:

short name	long name	short description
dr	**data_read**	Read of data, open for reading, etc.
dw	**data_write**	Write or modification of data
dc	**data_create**	Creation or deletion of any object
da	**data_access_change**	Change in object access (modes, owner)
lo	**login_logout**	Login, logout, creation by **at**(1)
ad	**administrative**	Normal administrative operation
p0	**minor_privilege**	Privileged operation
p1	**major_privilege**	Unusual privileged operation

EXAMPLE

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

```
dir: /etc/security/audit/jedgar/eggplant
dir: /etc/security/audit/jedgar.aux/eggplant
#
# Last-ditch audit file system when jedgar fills up.
#
dir: /etc/security/audit/global/eggplant
minfree: 20
flags: lo,p0,p1,ad,-all,^-da
```

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, privileged and administrative operations are to be audited (whether or not they succeed), and that failures of all types except failures to access data are to be audited.

FILES

/etc/security/audit/audit_control
/etc/security/audit/audit_warn
/etc/security/audit/*/*/*
/etc/security/passwd_adjunct

SEE ALSO

at(1), **audit**(2), **getfauditflags**(3), **audit.log**(5), **audit**(8), **auditd**(8)

NAME

audit_data – current information on audit daemon

SYNOPSIS

/etc/security/audit/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.

EXAMPLE

64:/etc/security/audit/auditserv/auditclient/2df0504

FILES

/etc/security/audit/audit_data

SEE ALSO

audit(2), **audit.log**(5), **audit**(8), **auditd**(8)

NAME

auto.home – automount map for home directories

SYNOPSIS

/etc/auto.home

AVAILABILTITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

auto.home resides in the /etc directory, and contains **automount**(8) map entries for user's home directories. On Sun386i systems, this file is used to build the **auto.home** Network Information Service (NIS) map used by **automount** at system startup and reads the **auto.master** NIS database, which contains an entry for **auto.home** and **/home** . The **auto.home** map contains entries for each username in the NIS **passwd** map, and the *hostname*:*/directory* to NFS mount.

References to **/home/***username* are translated by the automount daemon using the **auto.home** map, and the directory specified in the map entry is **nfs** mounted and that directory returned to the user's program.

User accounts created using **snap**(1) or **logintool**(8) have **passwd**(5) entries where the initial (home) directory name is, in the form **/home/***username*. **snap** and **logintool** also automatically create the **auto.home** entry for a user account. The format of the entry is described in **automount**(8). An example entry is:

mtravis system2:/export/home/users/mtravis

Thus, when the user **mtravis** logs into a Sun386i systems, the automounter automatically mounts his home directory from **system2**. This allows a user to log in to any Sun386i workstation on the network and be automatically placed in their home directory.

The convention for the format of home directory names used by **snap** and **logintool** is:

/export/home/*groupname/username*

Note: this is a different map and mechanism for home directories than the one that the automount daemon provides with the **–homes** switch. This is because the Sun386i convention for the format of home directory names differs and provides directories that can be used as mount points on a per user and per group basis.

FILES

/etc/auto.home

SEE ALSO

snap(1), **passwd**(5), **automount**(8), **logintool**(8)

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.

NAME

auto.vol – automount map for volumes

SYNOPSIS

/etc/auto.vol

AVAILABILTITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

auto.vol resides in the **/etc** directory, and contains **automount**(8) map entries for **volumes**. On Sun386i systems, this file is used to build the **auto.vol** Network Information Service (NIS) map used by **automount**(8) at system startup. **automouunt** reads the **auto.master** NIS map, which contains an entry for **auto.vol** and **/vol**.

References to **/vol/***volume_name* are translated by the automount daemon using the **auto.vol** map, and the directory specified in the map entry is mounted.

The concept of a volume is that it is a self contained directory hierarchy that can be NFS mounted. It is referenced using a known *volume_name*. The use of an automount map is suggested so that the volume and its contents can be referenced through **/vol**. This is advantageous because location-transparency (that is, which host the volume is on) and replication of read-only volumes can be provided using the automount mechanism. The format of the entry is described in **automount**(8). An example entry is:

archive system4:/export/archive

In the above example, the **archive** volume is currently on line on **system4**. Users and programs can reference it via **/vol/archive**. If for some reason the volume had to be moved to another system, **system2** for example, the network or system administrator simply edits the map entry for the archive volume and changes the hostname to **system2** and then rebuilds the NIS maps.

archive system2:/export/archive

Users and programs can continue to refer to the archive volume using **/vol/archive**, unaware that the volume was moved to another system.

FILES

/etc/auto.vol

SEE ALSO

automount(8)

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.

NAME

bar – tape archive file format

DESCRIPTION

bar(1), (the tape archive command) dumps several files into one, in a medium suitable for transportation. This format is not compatible with the format generated by **tar**(1).

A *bar tape* or file is a series of blocks. Each block is of size **TBLOCK.** A file on the tape is represented by a header block that describes the file, followed by zero or more blocks that give the contents of the file. At the end of the tape are two blocks filled with binary zeros, as an EOF indicator.

The blocks are grouped for physical I/O operations. Each group of *n* blocks (where *n* is set by the **b** keyletter on the **bar**(1) command line — default is 20 blocks) is written with a single system call; on nine-track tapes, the result of this write is a single tape record. The last group is always written at the full size, so blocks after the two zero blocks contain random data. On reading, the specified or default group size is used for the first read, but if that read returns less than a full tape block, the reduced block size is used for further reads, unless the **B** keyletter is used.

The header block looks like:

```
#define TBLOCK512

union hblock {
        char dummy[TBLOCK];
        struct header {
                char mode[8];
                char uid[8];
                char gid[8];
                char size[12];
                char mtime[12];
                char chksum[8];
                char rdev[8];
                char linkflag;
                char bar_magic[2];
                char volume_num[4];
                char compressed;
                char date[12];
                char start_of_name;
        } dbuf;
};
```

start_of_name is a null-terminated string. *date* is the date of the archive. *bar_magic* is a special number indicating that this is a **bar** archive. *rdev* is the device type, for files that are devices. The other fields are zero-filled octal numbers in ASCII. Each field (of width w) contains w-2 digits, a space, and a null, except *size*, *rdev*, and *mtime*, which do not contain the trailing null. *start_of_name* is the name of the file, as specified on the *bar* command line. Files dumped because they were in a directory that was named in the command line have the directory name as prefix and */filename* as suffix. *mode* is the file mode, with the top bit masked off. *uid* and *gid* are the user and group numbers that own the file. *size* is the size of the file in bytes. Links and symbolic links, and special files, are dumped with this field specified as zero. *mtime* is the modification time of the file at the time it was dumped. *chksum* is a decimal ASCII value that represents the sum of all the bytes in the header block. When calculating the checksum, the *chksum* field is treated as if it were all blanks. *linkflag* is ASCII 0 if the file is "normal" or a special file, 1 if it is an hard link, 2 if it is a symbolic link, and 3 if it is a special file (device or FIFO). The name linked-to, if any, is in a null-terminated string, following *start_of_name*. Unused fields of the header are binary zeros (and are included in the checksum).

The first time a given i-node number is dumped, it is dumped as a regular file. The second and subsequent times, it is dumped as a link instead. Upon retrieval, if a link entry is retrieved, but not the file it was linked to, an error message is printed and the tape must be manually re-scanned to retrieve the linked-to file.

When the H modifier is used with **bar** , an additional header block (one that does not pertain to a particular file) is written to the first block of each volume of the archive. The header ID, as specified on the command line, is copied to *start_of_name*. The size reflects the number of bytes to skip to the start of the first full file (always zero on the first volume).

The encoding of the header is designed to be portable across machines.

SEE ALSO

bar(1)

NAME

boards.pc – information about AT- and XT-compatible boards for DOS windows

SYNOPSIS

/etc/dos/defaults/boards.pc

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

The **boards.pc** file stores information about AT- and XT-compatible boards installed on a system.

Only the super-user may alter the file.

The file format is as follows, with entries separated by SPACE or TAB characters:

Board-name *I/O port range* *IRQ* *DMA* *Memory* *Options*

Board-name

The name of the board as it will appear in the DOS Windows Device menu. Use any name that is not longer than 19 characters.

I/O port range

Most boards have I/O addresses through which they exchange information with the workstation. For boards that will be used by DOS, the I/O address is entered in the **boards.pc** file, directly to the right of the board name.

Certain I/O addresses are already used by DOS Windows emulated devices (such as drive C and the DOS printers), and by built-in system hardware. The following list shows the AT-bus I/O address spaces:

Address	DOS Use
1F8-1FF *	**Hard disk (C:) emulation**
218-21F	**Expanded memory**
230-23F	**Bus mouse emulation**
278-27F	**Parallel port 2 (usually accessed through LPT3)**
378-37F *	**Parallel port 1 (usually accessed through LPT2)**
3B0-3BF	**Monochrome display adapter**
3D0-3DF	**Color display adapter**
3F0-3F7 *	**Diskette controller**

An address marked with an asterisk cannot be replaced by a board. When the board you are installing uses one of these addresses, or it uses the same address as another board that is already installed, change the jumpers or switch settings on your board to use a different address. If you add a board that occupies one of these address spaces, DOS ignores the entry. An address not marked with an asterisk may be used for a board you are installing, as long as you do not plan to use the emulated device at that address.

Adding an I/O Address Entry to boards.pc:

If the board uses addresses that can be contained within one eight-address block, note the block base address and include it in the *I/O port range* column of the **boards.pc** file. When using a multiple-block address, specify the base address of each block. For example, when entering a two-block address, specify the base addresses of both the first and second blocks, and separated with a SPACE character. Suppose you have a board with a two-block I/O address space that begins at 380. You would specify 380 388 in the **boards.pc** file's *I/O port range* column.

IRQ Some boards send periodic signals asking DOS to delay whatever it is doing and accept information from the device. These signals are known as **interrupt requests**, or more simply, as **interrupts**. The following chart shows the interrupt levels available under DOS Windows. Valid interrupt levels are 1 to 15, although some of these are reserved for emulated DOS devices.

Interrupt Level	Availability
0	**Unavailable; used for timer emulation**
1	**Unavailable; used for keyboard emulation**
2	**Unavailable; used for interrupt controller 2 cascade**
3	**Available for board, unless COM2 emulation in use (specified in setup.pc)**
4	**Available for board, unless COM1 emulation in use (specified in setup.pc)**
5	**Available for board, unless LPT3 emulation in use (specified in setup.pc)**
6	**Unavailable; used for diskette drive emulation**
7	**Unavailable; used by built-in parallel port**
8	**Unavailable; used for real-time clock emulation**
9	**Available for board**
10	**Available for board**
11	**Available for board**
12	**Available for board**
13	**Unavailable; used for 8087 numeric coprocessor emulation**
14	**Unavailable; used for hard disk emulation**
15	**Available for board**

To ensure that signals do not become confused, set each board or emulated device that uses interrupts for a different interrupt level. Normally, interrupt settings are changed by pressing small switches or moving metal jumpers on the board itself. Consult the manual of the board you are installing for details on how this is done. In addition to the changes required on the board itself, make sure that the interrupt level in your **boards.pc** file matches the setting on the card. For example, if a board's physical interrupt was previously 3, and you change it to 4 by altering switch settings or board jumpers, make a corresponding change in the **boards.pc** file. If the card uses a DOS driver, you may also need to make changes in **C:CONFIG.SYS** or other files to reflect the switch settings on the card.

Adding an Interrupt Entry to boards.pc

Some boards do not generate interrupts, and therefore will not have an interrupt level listed in their manuals. If this is the case, leave the *IRQ* column empty. For boards where an interrupt level is required, enter the letters **irq** followed by the appropriate number in the **boards.pc** file, as shown in EXAMPLES below.

DMA Certain boards use direct memory access (DMA) channels to ensure speedy transfer of large quantities of data. DMA channels 0, 1, 3, and 5 are available. Each DOS or SunOS DMA board on the system must be assigned a unique DMA channel. When two or more boards expect to use DMA channel 1, physically alter DMA settings on one of the boards so that it uses a different channel (such as DMA channel 3). Normally these settings are changed by pressing small switches or moving metal jumpers on the board itself. Consult the manual for the board you are installing for details on changing a DMA channel setting.

Adding a DMA Entry to boards.pc

When the board you are installing uses a DMA channel, include a **dma** entry for that board. For example, when the board is set up to use DMA channel 3, the entry can look like this:

MYBOARD 200 208 irq 2 dma 3

Memory

Some boards are equipped with memory chips for DOS. Because this memory is "mapped" (transferred) into DOS memory so that DOS can read it, the boards are called *memory mapped boards*. When you install such a board, include a **mem** entry with the following format:

mem *address size*

The *address* is the starting address of the memory segment, in hexadecimal notation. Enter the size of the memory block in kilobytes, in decimal notation. The following example is for a board that starts mapped memory at the address $DE00 and uses a block of 8 kilobytes.

MYBOARD 258 irq 5 dma 3 mem de00 8

When determining the size of the memory block, be careful not to confuse DOS address size (the number you should use) with actual on-board memory (the number you should not use). For example, a LIM memory board might have 2 megabytes of on-board memory, yet may require only 64 kilobytes of DOS address space for its memory mapping. Therefore, the number to use for the **mem** entry is 64.

Options

reboot

Certain boards require DOS rebooting before they work. These same boards require that you reboot DOS after you have finished using them. You can set up DOS to reboot the current DOS window automatically whenever the board is attached. DOS displays a confirmatory alert before rebooting.

To force DOS to reboot when you attach the board, add the word **reboot** at the end of the **boards.pc** line for that board, as shown in the following example:

MYBOARD 3e8 mem a000 192 reboot

If you choose to omit the **reboot** instruction, you can enable the board by attaching it and then manually rebooting:

1. Choose **Attached from the Device** menu to enable the board.
2. Choose **Reboot DOS Window.**

To detach such a board from a DOS window, choose **Detach** and then reboot the DOS window.

shared

You can specify that a device is to be shared between windows, rather than being reserved for use by one window at a time. Generally, you should do this only with devices, such as joysticks, which can fluidly move from one DOS window to another. To designate a device as shared, place the word **shared** at the very end of the **boards.pc** line:

Joystick 200 shared

Determining Board Information

In many cases, you may need to determine whether a board you are installing will conflict with other devices on the system. Also, you sometimes may need to install a board for which there is no entry in the **boards.pc** file. In most cases, the instruction manual included with the board you are installing should contain the technical information you need, including:

The I/O port addresses at which the board is accessed. One or more blocks can be reserved, and there are eight consecutive addresses per block.

The board's interrupt level, if the board generates interrupts.

The DMA channel number, if the board uses a direct memory access channel.

Memory mapping information, if the board maps data into DOS memory.

If the board's manual does not provide such information, contact the manufacturer.

EXAMPLES

The following is an example of a **boards.pc** file:

```
#COM2          2f8                          irq 3
#Joystick      200                                                  shared
#EGA           3b0 3b8 3c0 3c8 3d0 3d8                   mem a000 192 reboot
#VGA           3b0 3b8 3c0 3c8 3d0 3d8 102 2e8           mem a000 192
#3COM-3C501    300 308                      irq 3 dma 1
#TOPS-FlashTalk 398                         irq 3
#IBM-3363-Worm 258                          irq 5 dma 3 mem de00 8   reboot
#Plus-Hardcard20 320                        irq 5 dma 3 mem ca00 8   reboot
#HP-Basic      390                          irq 3
#DCA-IRMA1     220 228
#DCA-IRMA2     220 228 280 288
#Bernoulli-A220H 350                                                 reboot
#WD8003E       280 288 290 298              irq 5       mem d000 8
#NI5210        360                          irq 5       mem c000 16
#NIC                                        irq 5       mem d000 32
#LPT2          278                          irq 5
```

FILES

/usr/lib/help/*/*

SEE ALSO

dos(1), **setup.pc**(5)

Sun386i Advanced Skills

NAME

bootparams – boot parameter data base

SYNOPSIS

/etc/bootparams

DESCRIPTION

The **bootparams** file contains the list of client entries that diskless clients use for booting. For each diskless client the entry should contain the following information:

name of client
a list of keys, names of servers, and pathnames.

The first item of each entry is the name of the diskless client. The subsequent item is a list of keys, names of servers, and pathnames.

Items are separated by TAB characters.

A client entry in the local **/etc/bootparams** file supersedes an entry in the corresponding Network Information Service (NIS) map.

EXAMPLE

Here is an example of the **/etc/bootparams** taken from a SunOS system.

```
myclient        root=myserver:/nfsroot/myclient \
        swap=myserver:/nfsswap/myclient \
        dump=myserver:/nfsdump/myclient
```

FILES

/etc/bootparams

SEE ALSO

bootparamd(8)

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.

NAME

bootservers – NIS bootservers file

SYNOPSIS

/etc/bootservers

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

The **bootservers** file is an ASCII file that resides in the /etc directory on the Network Information Service (NIS) master server. The file contains basic information about each host providing boot services for clients on the network. This file contains a one-line entry for each boot server, where each field *must* be separated by a TAB character:

system type client_limit swap_size tmp_size root_minfree swap_minfree

The entries in the file have the following descriptions:

system	is the name of a boot server. This field contains only lowercase and numeric characters, must start with a lower-case character, and must not be longer than 32 characters.
type	Currently, the only legal value is 3.
client_limit	indicates the maximum number of diskless clients the server is willing to accept.
swap_size	default swap size per client (in kilobytes).
tmp_size	default tmp size per client (in kilobytes).
root_minfree	minimum amount of disk space in the server's client-root partition after a client is added (in kilobytes).
swap_minfree	minimum amount of disk space in the server's client-swap partition after a client is added (in kilobytes).

EXAMPLE

Here is a sample **bootservers** file entry:

```
polaris 3 2 16000 8000 40000 0
```

FILES

/etc/bootservers

SEE ALSO

System and Network Administration,
Sun386i Advanced Administration

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

coff – common assembler and link editor output

SYNOPSIS

#include <filehdr.h>
#include <aouthdr.h>
#include <scnhdr.h>
#include <reloc.h>
#include <linenum.h>
#include <storclass.h>
#include <syms.h>

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

The output from the link editor and the assembler (named **a.out** by default) is in COFF format (Common Object File Format) on the Sun386i system.

A common object file consists of a file header, a system header (if the file is link editor output), a table of section headers, a data section, relocation information, (optional) line numbers, a symbol table, and a string table. The general format looks like this:

file-header
system-header
section-headers
data
relocation
line-numbers
symbol-table
string-table

section-headers contains a number of section headers:

section 1 header
...
section n header

Similarly, *data*, *relocation*, and *line-numbers* are each divided into *n* sections.

The last three parts of an object file (line numbers, symbol table and string table) may be missing if the program was linked with the **–s** option of **ld**(1) or if they were removed by **strip**(1). Also note that the relocation information will be absent after linking unless the **–r** option of **ld**(1) was used. The string table exists only if the symbol table contains symbols with names longer than eight characters.

The sizes of each section (contained in the header, discussed below) are in bytes.

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 starts at location 0x1000 by default.

The **a.out** file produced by **ld**(1) has the magic number 0413 in the first field of the system header. The headers (file header, system header, and section headers) are loaded at the beginning of the text segment and the text immediately follows the headers in the user address space. The first text address will equal 0x1000 plus the size of the headers, and will vary depending upon the number of section headers in the **a.out** file. In an **a.out** file with three sections (.text, .data, .bss, and .comment), the first text address is at 0x000010D0. 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 4K boundary past the last text address. The first data address is determined by the following: If an **a.out** file were split into 4K chunks, one of the chunks would contain both the end of text and the beginning of data. When the **a.out** file is loaded into memory for execution, that chunk will appear twice; once at the end of text and once at the beginning of data (with some unused space in between). 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 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 segment boundary past the end of text plus the remainder of the last text address divided by 4K. If the last text address is a multiple of 4K no duplication is necessary.

On the Sun386i computer the stack begins at location 0xFBFFFFFF and grows toward lower addresses. The stack is automatically extended as required. The data segment is extended only as requested by the **brk**(2) system call.

For relocatable files the value of a word in the text or data portions that is not a reference to an undefined external symbol is exactly the value that will appear in memory when the file is executed. If a word in the text involves a reference to an undefined external symbol, there will be a relocation entry for the word, the storage class of the symbol-table entry for the symbol will be marked as an "external symbol", and the value and section number of the symbol-table entry will be undefined. When the file is processed by the link editor and the external symbol becomes defined, the value of the symbol will be added to the word in the file.

File Header

The format of the file header is:

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

System Header

The format of the system header is:

```
typedef struct aouthdr
{
        short   magic;          /* magic number */
        short   vstamp;         /* version stamp */
        long    tsize;          /* text size in bytes, padded */
        long    dsize;          /* initialized data (.data) */
        long    bsize;          /* uninitialized data (.bss) */
        long    entry;          /* entry point */
        long    text_start;     /* base of text used for this file */
        long    data_start;     /* base of data used for this file */
} AOUTHDR;
```

Section Header

The format of the section header is:

```
struct scnhdr
{
        char            s_name[SYMNMLEN];/* section name */
        long            s_paddr;  /* physical address */
        long            s_vaddr;  /* virtual address */
        long            s_size;   /* section size */
        long            s_scnptr; /* file ptr to raw data */
        long            s_relptr; /* file ptr to relocation */
        long            s_lnnoptr;/* file ptr to line numbers */
        unsigned shorts_nreloc;  /* # reloc entries */
        unsigned shorts_nlnno;   /* # line number entries */
        long            s_flags;  /* flags */
};
```

Relocation

Object files have one relocation entry for each relocatable reference in the text or data. If relocation information is present, it will be in the following format:

```
struct reloc
{
        long        r_vaddr;  /* (virtual) address of reference */
        long        r_symndx;/* index into symbol table */
        ushort      r_type;   /* relocation type */
};
```

The start of the relocation information is **s_relptr** from the section header. If there is no relocation information, **s_relptr** is 0.

Line Number

The cc(1V) command generates an entry in the object file for each C source line on which a breakpoint is possible (when invoked with the **–g** option. Users can refer to line numbers when using the appropriate debugger, such as **dbx**(1)). The structure of these line number entries appears below.

```
struct lineno
{
        union
        {
                long    l_symndx ;
                long    l_paddr ;
        }               l_addr ;
        unsigned shortl_lnno ;
} ;
```

Numbering starts with one at the top of the source file and increments independent of transition between functions. The initial line number entry for a function has **l_lnno** equal to zero, and the symbol table index of the function's entry is in **l_symndx**. Otherwise, **l_lnno** is non-zero, and **l_paddr** is the physical address of the code for the referenced line. Thus the overall structure is the following:

l_addr	l_lnno
function symtab index	0
physical address	line
physical address	line
...	
function symtab index	0

```
physical address        line
physical address        line
...
```

Symbol Table

The format of each symbol in the symbol table is described by the syment structure, shown below. This structure is compatible with System V COFF, but has an added _n_dbx structure which is needed by **dbx**(1).

```
#define SYMNMLEN 8
#define FILNMLEN 14
#define DIMNUM   4

struct syment
{
  union                         /* all ways to get a symbol name */
  {
    char        _n_name[SYMNMLEN]; /* name of symbol */
    struct
    {
      long      _n_zeroes;      /* == 0L if in string table */
      long      _n_offset;      /* location in string table */
    } _n_n;
    char        *_n_nptr[2];    /* allows overlaying */
    struct
    {
      char      _n_leading_zero; /* null char */
      char      _n_dbx_type;    /* stab type */
      short     _n_dbx_desc;    /* value of desc field */
      long      _n_stab_ptr;    /* table ptr */
    } _n_dbx;
  } _n;
  long          n_value;        /* value of symbol */
  short         n_scnum;        /* section number */
  unsigned short n_type;        /* type and derived type */
  char          n_sclass;       /* storage class */
  char          n_numaux;       /* number of aux entries */
};

#define n_name      _n._n_name
#define n_zeroes    _n._n_n._n_zeroes
#define n_offset    _n._n_n._n_offset
#define n_nptr      _n._n_nptr[1]
```

The storage class member (**n_sclass**) is set to one of the constants defined in **<storclass.h>**. Some symbols require more information than a single entry; they are followed by *auxiliary entries* that are the same size as a symbol entry. The format follows:

```
union auxent {
        struct {
                long    x_tagndx;
                union {
                        struct {
                                unsigned short  x_lnno;
                                unsigned short  x_size;
                        } x_lnsz;
                        long    x_fsize;
                } x_misc;
                union {
                        struct {
                                long    x_lnnoptr;
                                long    x_endndx;
                        } x_fcn;
                        struct {
                                unsigned short  x_dimen[DIMNUM];
                        } x_ary;
                } x_fcnary;
                unsigned short  x_tvndx;
        } x_sym;

        struct {
                char    x_fname[FILNMLEN];
        } x_file;

        struct {
                long        x_scnlen;
                unsigned short  x_nreloc;
                unsigned short  x_nlinno;
        } x_scn;

        struct {
                long            x_tvfill;
                unsigned short  x_tvlen;
                unsigned short  x_tvran[2];
        } x_tv;
};
```

Indexes of symbol table entries begin at *zero*. The start of the symbol table is **f_symptr** (from the file header) bytes from the beginning of the file. If the symbol table is stripped, **f_symptr** is 0. The string table (if one exists) begins at **f_symptr** + (**f_nsyms** * SYMESZ) bytes from the beginning of the file.

SEE ALSO

as(1), **cc**(1V), **ld**(1), **brk**(2), **ldfcn**(3)

NAME

core – format of memory image file

SYNOPSIS

#include <sys/core.h>

DESCRIPTION

The operating system writes out a memory image of a terminated process when any of various errors occur. See **sigvec**(2) for the list of reasons; the most common are memory violations, illegal instructions, bus errors, and user-generated quit signals. The memory image is called **core** and is written in the process's working directory (provided it can be; normal access controls apply). Set-user-ID and set-group-ID programs do not produce core files when they terminate as this would cause a security loophole.

The maximum size of a **core** file is limited by **setrlimit** (see **getrlimit**(2)). Files which would be larger than the limit are not created.

The core file consists of a **core** structure, as defined in the **<sys/core.h>** file, followed by the data pages and then the stack pages of the process image. The **core** structure includes the program's header, the size of the text, data, and stack segments, the name of the program and the number of the signal that terminated the process. The program's header is described by the **exec** structure defined in the **<sys/exec.h>** file, except on Sun386i systems.

```
struct core {
        int     c_magic;        /* Corefile magic number */
        int     c_len;          /* Sizeof (struct core) */
        struct  regs c_regs;    /* General purpose registers */
        struct  exec c_aouthdr; /* A.out header */
        int     c_signo;        /* Killing signal, if any */
        int     c_tsize;        /* Text size (bytes) */
        int     c_dsize;        /* Data size (bytes) */
        int     c_ssize;        /* Stack size (bytes) */
        char    c_cmdname[CORE_NAMELEN + 1]; /* Command name */
        struct  fpu c_fpu;      /* external FPU state */
        int     c_ucode;        /* Exception no. from u_code */
};
```

The members of the structure are:

c_magic The magic number CORE_MAGIC , as defined in **<sys/core.h>**.

c_len The length of the **core** structure in the core file. This need not be equal to the current size of a **core** structure as defined in **<sys/core.h>**, as the core file may have been produced on a different release of the SunOS operating system.

c_regs The general purpose registers at the time the core file was produced. This structure is machine-dependent.

c_aouthdr The executable image header of the program.

c_signo The number of the signal that terminated the process; see **sigvec**(2).

c_tsize The size of the text segment of the process at the time the core file was produced.

c_dsize The size of the data segment of the process at the time the core file was produced. This gives the amount of data space image in the core file.

c_ssize The size of the stack segment of the process at the time the core file was produced. This gives the amount of stack space image in the core file.

c_cmdname The first CORE_NAMELEN characters of the last component of the path name of the program.

c_fpu The status of the floating point hardware at the time the core file was produced.

c_ucode The signal code of the signal that terminated the process, if any. See **sigvec**(2).

SEE ALSO

adb(1), **dbx**(1), **getrlimit**(2), **sigvec**(2)

NAME

cpio – format of cpio archive

DESCRIPTION

The old format *header* structure, when the **–c** option of **cpio** is not used, is:

```
struct {
        short   h_magic,
                h_dev;
        ushort  h_ino,
                h_mode,
                h_uid,
                h_gid;
        short   h_nlink,
                h_rdev,
                h_mtime[2],
                h_namesize,
                h_filesize[2];
        char    h_name[h_namesize rounded to a word];
} Hdr;
```

The byte order here is that of the machine on which the tape was written. If the tape is being read on a machine with a different byte order, you have to use **swab**(3) after reading the header. You can determine what byte order the tape was written with by examining the *h_magic* field; if it is equal to 0143561 (octal), which is the standard magic number 070707 (octal) with the bytes swapped, the tape was written in a byte order opposite to that of the machine on which it is being read. If you are producing a tape to be read on a machine with the opposite byte order to that of the machine on which it is being produced, you can use **swap** before writing the header.

When the **–c** option is used, the *header* information is described by the statement below:

```
sscanf(Chdr, "%6o%6o%6o%6o%6o%6o%6o%6o%11lo%6o%11lo%s",
        &Hdr.h_magic, &Hdr.h_dev, &Hdr.h_ino, &Hdr.h_mode,
        &Hdr.h_uid, &Hdr.h_gid, &Hdr.h_nlink, &Hdr.h_rdev,
        &Hdr.h_mtime, &Hdr.h_namesize, &Hdr.h_filesize, &Hdr.h_name);
```

Longtime and *Longfile* are equivalent to *Hdr.h_mtime* and *Hdr.h_filesize*, respectively. The contents of each file is recorded in an element of the array of varying length structures, *archive*, together with other items describing the file. Every instance of *h_magic* contains the constant 070707 (octal). The items *h_dev* through *h_mtime* have meanings explained in **stat**(2V). The length of the null-terminated path name *h_name*, including the null byte, is given by *h_namesize*.

The last record of the *archive* always contains the name **TRAILER!!!**. Special files, directories, and the trailer, are recorded with *h_filesize* equal to zero. Symbolic links are recorded similarly to regular files, with the ''contents'' of the file being the name of the file the symbolic link points to.

SEE ALSO

cpio(1), **find**(1), stat(2V), **swab**(3)

NAME

crontab – table of times to run periodic jobs

SYNOPSIS

/var/spool/cron/crontabs/*

DESCRIPTION

The **cron** utility is a permanent process, started by **/etc/rc.local**. **cron** consults the files in the directory **/var/spool/cron/crontabs** to find out what tasks are to be done, and at what time.

Each line in a **crontab** file consists of six fields, separated by spaces or tabs, as follows:

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

minutes	Minutes field, which can have values in the range 0 through 59.
hours	Hours field, which can have values in the range 0 through 23.
day-of-month	Day of the month, in the range 1 through 31.
month	Month of the year, in the range 1 through 12.
day-of-week	Day of the week, in the range 0 through 6. Sunday is day 0 in this scheme of things. For backward compatibility with older systems, Sunday may also be specified as day 7.
command	Command to be run. A percent character in this field (unless escaped by \) is translated to a NEWLINE character. Only the first line (up to a % or end of line) of the command field is executed by the Shell. The other lines are made available to the command as standard input.

Any of fields 1 through 5 can be a list of values separated by commas. A value can either be a number, or a pair of numbers separated by a hyphen, indicating that the job is to be done for all the times in the specified range. If a field is an asterisk character (*) it means that the job is done for all possible values of the field.

Note: the specification of days may be made by two fields (day of the month and day of the week). If both are specified as a list of elements, both are adhered to. For example,

0 0 1,15 * 1

would run a command on the first and fifteenth of each month, as well as on every Monday. To specify days by only one field, the other field should be set to *. For example,

0 0 * * 1

would run a command only on Mondays.

The command is run from your home directory with an **arg0** of **sh**. Users who desire to have their **.profile** executed must explicitly do so in the command. **cron** supplies a default environment for every shell, defining **HOME, LOGNAME, USER, SHELL(=/bin/sh)**, and **PATH(=:/usr/ucb:/bin:/usr/bin)**.

NOTE: Users should remember to redirect the standard output and standard error of their commands! If this is not done, any generated output or errors will be mailed to the user.

Lines that start with # are treated as comments.

EXAMPLES

```
0 0 * * * calendar –
15 0 * * * /usr/etc/sa –s >/dev/null
15 4 * * * find /var/preserve –mtime +7 –a –exec rm –f {} ;
40 4 * * * find / –name '#*' –atime +3 –exec rm –f {} ;
0 0 * * 1-5 /usr/local/weekdays
0 0 * * 0,6 /usr/local/weekends
```

The **calendar** command runs at minute 0 of hour 0 (midnight) of every day. The **/usr/etc/sa** command runs at 15 minutes after midnight every day. The two **find** commands run at 15 minutes past four and at 40 minutes past four, respectively, every day of the year. The **/usr/local/weekdays** command is run at midnight on weekdays. Finally, the **/usr/local/weekends** command is run at midnight on weekends.

FILES

/var/spool/cron/crontabs/*
tables of times to run periodic jobs

/etc/rc.local

.profile

SEE ALSO

cron(8), **rc**(8)

NAME

dir – format of directories

SYNOPSIS

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

DESCRIPTION

A directory behaves exactly like an ordinary file, save that no user may write into a directory and directories must be read using the **getdirentries**(2) system call or the **directory**(3V) library routines. The fact that a file is a directory is indicated by a bit in the flag word of its inode entry; see **fs**(5).

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 (512 bytes on most machines):

```
#ifdef KERNEL
#define DIRBLKSIZ DEV_BSIZE
#else
#define DIRBLKSIZ 512
#endif

#define MAXNAMLEN 255
```

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 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.

The macro **DIRSIZ(dp)** gives the amount of space required to represent a directory entry. Free space in a directory is represented by entries that have:

```
dp->d_reclen > DIRSIZ(dp)
```

All DIRBLKSIZ bytes in a directory block are claimed by the directory entries. This usually results in the last entry in a directory having a large **dp->d_reclen**. When entries are deleted from a directory, the space is returned to the previous entry in the same directory block by increasing its **dp->d_reclen**. If the first entry of a directory block is free, then its **dp->d_ino** is set to 0. Entries other than the first in a directory do not normally have **dp->d_ino** set to 0.

The **DIRSIZ** macro gives the minimum record length which will hold the directory entry. This requires the amount of space in struct direct without the **d_name** field, plus enough space for the name with a terminating null byte (**dp->d_namlen+1**), rounded up to a 4-byte boundary.

```
#undef DIRSIZ
#define DIRSIZ(dp)    ((sizeof (struct direct) - (MAXNAMLEN+1)) + (((dp)->d_namlen+1 + 3) &~ 3))
struct   direct {
        u_long  d_ino;
        short   d_reclen;
        short   d_namlen;
        char    d_name[MAXNAMLEN + 1];
        /* typically shorter */
};
```

By convention, the first two entries in each directory are for '.' and '..'. The first is an entry for the directory itself. The second is for the parent directory. The meaning of '..' is modified for the root directory of the master file system ("/"), for which '..' has the same meaning as '.'.

SEE ALSO

getdirentries(2), **directory**(3V), **fs**(5)

NAME

dump, dumpdates – incremental dump format

SYNOPSIS

```
#include <sys/types.h>
#include <sys/inode.h>
#include <protocols/dumprestore.h>
```

DESCRIPTION

Tapes used by **dump** and **restore**(8) 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 TS_TAPE             1
#define TS_INODE            2
#define TS_BITS             3
#define TS_ADDR             4
#define TS_END              5
#define TS_CLRI             6
#define OFS_MAGIC           (int)60011
#define NFS_MAGIC           (int)60012
#define CHECKSUM            (int)84446
union u_spcl {
        char dummy[TP_BSIZE];
        struct                  s_spcl {
                                intc_type;
                                time_tc_date;
                                time_tc_ddate;
                                intc_volume;
                                daddr_tc_tapea;
                                ino_tc_inumber;
                                intc_magic;
                                intc_checksum;
                                structdinodec_dinode;
                                intc_count;
                                charc_addr[TP_NINDIR];
        } s_spcl;
} u_spcl;

#define spcl u_spcl.s_spcl

#define DUMPOUTFMT          "%-16s %c %s"/* for printf */
                            /* name, incno, ctime(date) */
#define DUMPINFMT           "%16s %c %[^\n]\n"/* inverse for scanf */
```

TP_BSIZE Size of file blocks on the dump tapes. Note: **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 **dump**.

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 two.

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_BITS A bit map follows. This bit map has a one bit for each inode that was dumped.

TS_ADDR A subrecord of a file description. See **c_addr** below.

TS_END End of tape record.

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.

NFS_MAGIC All header records have this number in **c_magic**.

CHECKSUM Header records checksum to this value.

The fields of the header structure are as follows:

c_type The type of the header.

c_date The date the dump was taken.

c_ddate The date the file system was dumped from.

c_volume The current volume number of the dump.

c_tapea The current number of this (1024-byte) 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; see **fs**(5).

c_count The count of characters in **c_addr**.

c_addr An array of characters describing the blocks of the dumped file. A character is zero if the block associated with that character was not present on the file system, otherwise the character is non-zero. If the block was not present on the file system, no block was dumped; the block will be restored as a hole in the file. If there is not sufficient space in this record to describe all of the blocks in a file, **TS_ADDR** records will be scattered through the file, each one picking up where the last left off.

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 **dump**(8).

The date of the incremental dump in the format generated by **ctime**(3V).

DUMPOUTFMT is the format to use when using **printf**(3S) to write an entry to **/etc/dumpdates**; **DUMPINFMT** is the format to use when using **scanf**(3S) to read an entry from **/etc/dumpdates**.

FILES

/etc/dumpdates

SEE ALSO

fs(5), **types**(5), **dump**(8), **restore**(8)

NAME

environ – user environment

SYNOPSIS

extern char **environ;

DESCRIPTION

An array of strings called the 'environment' is made available by **execve**(2V) when a process begins. By convention these strings have the form '*name=value*'. The following names are used by various commands:

PATH The sequence of directory prefixes that **sh**(1), **time**(1V), **nice**(1), etc., apply in searching for a file known by an incomplete path name. The prefixes are separated by ':'. The **login**(1) process sets **PATH=:/usr/ucb:/bin:/usr/bin.**

HOME The name of the user's login directory, set by **login**(1) from the password file **/etc/passwd** (see **passwd**(5)).

TERM The type of terminal on which the user is logged in. This information is used by commands, such as **nroff**(1) or **plot**(1G), which may exploit special terminal capabilities. See **/etc/termcap** (**termcap**(5)) for a list of terminal types.

SHELL The path name of the user's login shell.

TERMCAP The string describing the terminal in TERM, or the name of the **termcap** file, see **termcap**(3X), **termcap**(5).

EXINIT A startup list of commands read by **ex**(1), **edit**, and **vi**(1).

USER
LOGNAME The user's login name.

TZ The name of the time zone that the user is located in. If **TZ** is not present in the environment, the system's default time zone, normally the time zone that the computer is located in, is used.

Further names may be placed in the environment by the *export* command and '*name=value*' arguments in **sh**(1), or by the **setenv** command if you use **csh**(1). Arguments may also be placed in the environment at the point of an **execve**(2V). It is unwise to conflict with certain **sh**(1) variables that are frequently exported by **.profile** files: **MAIL, PS1, PS2, IFS.**

SYSTEM V DESCRIPTION

The description of the variable **TERMCAP** does not apply to programs built in the System V environment.

FILES

/etc/passwd
etc/termcap

SEE ALSO

csh(1), **ex**(1), **login**(1), **nice**(1), **nroff**(1), **plot**(1G), **sh**(1), **time**(1V), **vi**(1), **execve**(2V), **getenv**(3V), **system**(3), **termcap**(3X), **passwd**(5), **termcap**(5)

NAME

ethers – Ethernet address to hostname database or NIS domain

DESCRIPTION

The **ethers** file contains information regarding the known (48 bit) Ethernet addresses of hosts on the Internet. For each host on an Ethernet, a single line should be present with the following information:

Ethernet-address official-host-name

Items are separated by any number of blanks 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 a SPACE, TAB, NEWLINE, or comment character. It is intended that host names in the **ethers** file correspond to the host names in the **hosts**(5) file.

The **ether_line**( ) routine from the Ethernet address manipulation library, **ethers**(3N) may be used to scan lines of the **ethers** file.

FILES

/etc/ethers

SEE ALSO

ethers(3N), **hosts**(5)

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.

NAME

exports, xtab – directories to export to NFS clients

SYNOPSIS

/etc/exports

/etc/xtab

DESCRIPTION

The **/etc/exports** file contains entries for directories that can be exported to NFS clients. This file is read automatically by the **exportfs**(8) command. If you change this file, you must run **exportfs**(8) for the changes to affect the daemon's operation.

Only when this file is present at boot time does the **rc.local** script execute **exportfs**(8) and start the NFS file-system daemon, **nfsd**(8).

The **/etc/xtab** file contains entries for directories that are *currently* exported. This file should only be accessed by programs using **getexportent** (see **exportent**(3)). (Use the **–u** option of **exportfs** to remove entries from this file).

An entry for a directory consists of a line of the following form:

directory *–option*[,*option*]...

directory is the pathname of a directory (or file).

option is one of

ro Export the directory read-only. If not specified, the directory is exported read-write.

rw=*hostnames*[:*hostname*]...
Export the directory read-mostly. Read-mostly means read-only to most machines, but read-write to those specified. If not specified, the directory is exported read-write to all.

anon=*uid*
If a request comes from an unknown user, use *uid* as the effective user ID. Note: root users (uid 0) are always considered "unknown" by the NFS server, unless they are included in the "root" option below. The default value for this option is –2. Setting "anon" to –1 disables anonymous access. Note: by default secure NFS will accept insecure requests as anonymous, and those wishing for extra security can disable this feature by setting "anon" to –1.

root=*hostnames*[:*hostname*]...
Give root access only to the root users from a specified *hostname*. The default is for no hosts to be granted root access.

access=*client*[:*client*]...
Give mount access to each *client* listed. A *client* can either be a hostname, or a netgroup (see **netgroup**(5)). Each *client* in the list is first checked for in the netgroup database, and then the hosts database. The default value allows any machine to mount the given directory.

secure Require clients to use a more secure protocol when accessing the directory.

A '#' (pound-sign) anywhere in the file indicates a comment that extends to the end of the line.

EXAMPLE

```
/usr        –access=clients          # export to my clients
/usr/local  # export to the world
/usr2       –access=hermes:zip:tutorial  # export to only these machines
/usr/sun    –root=hermes:zip         # give root access only to these
/usr/new    –anon=0                  # give all machines root access
```

```
/usr/bin        -ro                         # export read-only to everyone
/usr/stuff      -access=zip,anon=-3,ro      # several options on one line
```

FILES

/etc/exports
/etc/xtab
/etc/hosts
/etc/netgroup
rc.local

SEE ALSO

exportent(3), **hosts**(5), **netgroup**(5), **exportfs**(8), **nfsd**(8)

WARNINGS

You cannot export either a parent directory or a subdirectory of an exported directory that is *within the same filesystem*. It would be illegal, for instance, to export both **/usr** and **/usr/local** if both directories resided on the same disk partition.

NAME

ext_ports – external ports file for network printers, terminals, and modems

SYNOPSIS

/etc/ext_ports

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

The **ext_ports** external ports file is an ASCII file in the **/etc** directory on the Network Information Service (NIS) master server. **ext_ports** is used only by SNAP, and contains basic information about each printer, terminal, and modem on the network. This file contains a one-line entry for each device, and each field *must* be separated by a TAB character:

system:port type status baud model name #comment

system names the system to which the device is attached. This field contains only lower case and numeric characters, must start with a lower case character, and must not be longer than 32 characters.

port names the port in **/dev** on the *system*: **ttya** for the Sun386i serial port, **pp0** for the parallel port, and **ttym0** and **ttym1** for ports on an AT bus serial card.

type **printer**, **terminal**, or **modem**.

status indicates the device status. For terminals and printers, this can be **on** or **off**. An **off** status means the device is disabled from access by the SunOS operating system, but can still be accessed by DOS. For modems, this can be **in** to enable dialin, **out** to enable dialout, **in_out** to enable dialin and dialout, or **off**. An **off** status means the device is disabled from access by the SunOS operating system, but it can still be accessed by DOS.

baud is the baud rate.

model indicates the manufacturer or kind of device. For printers, this can be **epson**, **hp**, or **text**, for Epson and compatibles, HP Laserjet and compatibles, or for text-only printers. For terminals, this can be **vt100** or **wyse-50** for DEC VT-100 and compatibles or for Wyse WY-50 and compatibles. For modems, this can be **hayes** for Hayes and compatibles.

name is only used for unique naming of printers on the network. Up to 16 characters can be entered. This field is blank for terminals and modems — simply insert a TAB character.

#comment

can contain anything you want, up to a maximum of 96 characters.

EXAMPLE

In this example of an **ext_ports** file, the system vulcan has an epson printer attached to its parallel port, and a Wyse-50 terminal attached to its serial port, but with logins currently disabled. The system android has a VT100 attached to its serial port, with logins enabled. The system polaris has a **hayes** modem set for dialing out on an installed AT bus serial card.

```
vulcan:pp0      printer   on      9600   epson     lp   #Engineering lab
android:ttya    terminal  on      9600   vt100          #Reception
vulcan:ttya     terminal  off     9600   wyse-50        #Engineering lab
polaris:ttym0   modem     in_out  2400   hayes          #QA lab
```

FILES

/etc/ext_ports

SEE ALSO

snap(1), **vipw**(8)

Sun386i System and Network Administration,
Sun386i Advanced Administration

BUGS

The **/etc/ext_ports** file must be locked against simultaneous changes when it is edited; **vipw**(8) does the necessary locking.

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.

NAME

fbtab – framebuffer table

SYNOPSIS

/etc/fbtab

DESCRIPTION

The **/etc/fbtab** file contains information that is used by **login**(1), **getty**(8) and the window system (for example, **sunview**(1)) to change the owner, group, and permissions of window system devices upon logging into or out of a console device. By default, *all* lines in this file are commented out. That is, all window security is disabled. To enable window security, edit **/etc/fbtab**, log out, and log back in. You *must* edit this file before window security can be enabled.

The owner and group of the devices listed in **/etc/fbtab** are set to the owner and group of the console. The permissions are set as specified in **/etc/fbtab**. As in the example below, **0600** is the recommended permissions for normal security.

Fields are separated by TAB and/or SPACE characters. Blank lines and comments can appear anywhere in the file; comments are delimited by '#' and a NEWLINE.

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 (the full pathname is required).

Once the devices are owned by the user, their permissions and ownership can be changed using **chmod**(1V) and **chown**(8), as with any other user-owned file.

EXAMPLES

The following example entry in the **/etc/fbtab** file enables normal window security:

```
/dev/console 0600     /dev/kbd:/dev/mouse
/dev/console 0600     /dev/fb:/dev/bwone0:/dev/bwtwo0
/dev/console 0600     /dev/cgone0:/dev/cgtwo0:/dev/cgthree0:/dev/cgfour0
/dev/console 0600     /dev/cgsix0:/dev/cgeight0:/dev/cgnine0
/dev/console 0600     /dev/gpone0a:/dev/gpone0b:/dev/gpone0c:/dev/gpone0d
```

This entry specifies that upon login to **/dev/console**, the owner, group and permissions of all supported devices will be set to the user's username, the user's group and **0600**, respectively. You need only specify the devices supported by your configuration. Upon logout, the owner and group of these devices will be reset to **root** and **wheel**. The permissions remain as set in the **/etc/fbtab** file.

SEE ALSO

login(1), **sunview**(1), **sv_acquire**(1), **getty**(8)

NAME

fcntl – file control options

SYNOPSIS

```
#include <fcntl.h>
```

DESCRIPTION

The **fcntl**(2V) function provides for control over open files. This include file describes *requests* and *arguments* to **fcntl** and **open**(2V) as shown below:

```
/*              @ (#)fcntl.h 1.2 83/12/08 SMI; from UCB 4.2 83/09/25*/
/*
* Flag values accessible to open(2V) and fcntl(2)
*  (The first three can only be set by open)
*/
#define         O_RDONLY                0
#define         O_WRONLY                1
#define         O_RDWR                  2
#define         O_NDELAY                FNDELAY         /* Non-blocking I/O */
#define         O_APPEND                FAPPEND         /* append (writes guaranteed at the end) */
#ifndef         F_DUPFD
/* fcntl(2) requests */
#define         F_DUPFD                 0               /* Duplicate fildes */
#define         F_GETFD                 1               /* Get fildes flags */
#define         F_SETFD                 2               /* Set fildes flags */
#define         F_GETFL                 3               /* Get file flags */
#define         F_SETFL                 4               /* Set file flags */
#define         F_GETOWN                5               /* Get owner */
#define         F_SETOWN                6               /* Set owner */
/* flags for F_GETFL, F_SETFL— copied from <sys/file.h> */
#define         FNDELAY                                 00004/* non-blocking reads */
#define         FAPPEND                                 00010/* append on each write */
#define         FASYNC                                  00100/* signal pgrp when data ready */
#endif
```

SEE ALSO

fcntl(2V), **open**(2V)

NAME

fs, inode – format of a 4.2 (ufs) file system volume

SYNOPSIS

#include <sys/types.h>
#include <ufs/fs.h>
#include <ufs/inode.h>

DESCRIPTION

Standard 4.2 (ufs) file system storage volumes have a common format for certain vital information. Every such 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 include file **<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 contains inodes and data.

A file system is described by its super-block, which in turn describes the cylinder groups. The super-block is critical data and is replicated in each cylinder group 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 further unless disaster strikes.

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 of exclusively large data blocks. To avoid undue wasted disk space, the last data block of a small file is allocated as 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; to determine block availability, aligned fragments are examined.

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**(8).

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. This 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 4, thus the default fragment size is a fourth of the block size.

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

fs_rotdelay gives the minimum number of milliseconds to initiate another disk transfer on the same cylinder. It is used in determining the rotationally optimal layout for disk blocks within a file; the default value for **fs_rotdelay** is 2ms.

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 big 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.

inode: 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 include file **<ufs/inode.h>**.

SEE ALSO

mkfs(8)

NAME

fspec – format specification in text files

DESCRIPTION

It is sometimes convenient to maintain text files on the operating system with non-standard tab stop settings, (that is, tab stops that are not set at every eighth column). Such files must generally be converted to a standard format, frequently by replacing all TAB characters with the appropriate number of SPACE characters, before they can be processed by operating system commands. A format specification occurring in the first line of a text file specifies how TAB characters 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 stop settings for the file. The value of *tabs* must be one of the following:

- A list of column numbers separated by commas, indicating tab stops set at the specified columns;
- A '–' followed immediately by an integer *n*, indicating tab stops set at intervals of *n* columns, that is, at 1+*n*, 1+2*n*, and so on;
- A '–' followed by the name of a "canned" tab stop specification.

Up to 40 numbers are allowed in a comma-separated list of tab stop settings. If any number (except the first one) is preceded by a plus sign, it is taken as an increment to be added to the previous value. Thus, the formats **t1, 10, 20, 30** and **t1, 10, +10, +10** are considered identical.

Standard tab stops are specified by **t–8**, or equivalently, **t1, 9, 17, 25**, etc. This is the tab stop setting that most operating system utilities assume, and is the most likely setting to be found at a terminal. The specification **t–0** specifies no tab stops at all.

The "canned" tab stops specifications that are recognized are as follows:

a 1, 10, 16, 36, 72
Assembler, IBM S/370, first format

a2 1, 10, 16, 40, 72
Assembler, IBM S/370, second format

c 1, 8, 12, 16, 20, 55
COBOL, normal format

c2 1, 6, 10, 14, 49
COBOL compact format (columns 1-6 omitted). Using this code, the first typed character corresponds to card column 7, one space gets you to column 8, and a TAB reaches column 12. Files using this tab stop setup should include a format specification as follows:

<:t–c2 m6 s66 d:>

c3 1, 6, 10, 14, 18, 22, 26, 30, 34, 38, 42, 46, 50, 54, 58, 62, 67
COBOL compact format (columns 1-6 omitted), with more tab stops than **c2**. This is the recommended format for COBOL. The appropriate format specification is:

<:t–c3 m6 s66 d:>

f 1, 7, 11, 15, 19, 23
FORTRAN

p 1, 5, 9, 13, 17, 21, 25, 29, 33, 37, 41, 45, 49, 53, 57, 61
PL/I

s 1, 10, 55
SNOBOL

u 1, 12, 20, 44
UNIVAC 1100 Assembler

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

m *margin*
The **m** parameter specifies a number of SPACE characters 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), **tabs**(1V)

NAME

fstab, mtab – static filesystem mounting table, mounted filesystems table

SYNOPSIS

/etc/fstab

/etc/mtab

DESCRIPTION

The **/etc/fstab** file contains entries for filesystems and disk partitions to mount using the **mount**(8) command, which is normally invoked by the **rc.boot** script at boot time. This file is used by various utilities that mount, unmount, check the consistency of, dump, and restore file systems. It is also used by the system itself when locating the swap partition.

The **/etc/mtab** file contains entries for filesystems *currently* mounted, and is read by programs using the routines described in **getmntent**(3). **umount** (see **mount**(8)) removes entries from this file.

Each entry consists of a line of the form:

filesystem directory type options freq pass

filesystem is the pathname of a block-special device, the name of a remote filesystem in *host*:*pathname* form, or the name of a "swap file" made with **mkfile**(8).

directory is the pathname of the directory on which to mount the filesystem.

type is the filesystem type, which can be one of:

- **4.2** to mount a block-special device
- **lo** to loopback-mount a file system
- **nfs** to mount an exported NFS filesystem
- **swap** to indicate a swap partition
- **ignore** to have the **mount** command ignore the current entry (good for noting disk partitions that are not being used)
- **rfs** to mount an RFS filesystem
- **tmp** filesystem in virtual memory
- **hsfs** to mount an ISO 9660 Standard or High Sierra Standard CD-ROM filesystem

options contains a comma-separated list (no spaces) of mounting options, some of which can be applied to all types of filesystems, and others which only apply to specific types.

4.2 options:

- **quota | noquota** Disk quotas are enforced or not enforced. The default is **noquota**.

nfs options:

- **bg | fg** If the first attempt fails, retry in the background, or, in the foreground.
- **noquota** Prevent **quota**(1) from checking whether the user is over quota on this file system; if the file system has quotas enabled on the server, quotas will still be checked for operations on this file system.
- **retry=***n* The number of times to retry the mount operation.
- **rsize=***n* Set the read buffer size to *n* bytes.
- **wsize=***n* Set the write buffer size to *n* bytes.
- **timeo=***n* Set the NFS timeout to *n* tenths of a second.
- **retrans=***n* The number of NFS retransmissions.
- **port=***n* The server IP port number.
- **soft | hard** Return an error if the server does not respond, or continue the retry request until the server responds.
- **intr** Allow keyboard interrupts on hard mounts.
- **secure** Use a more secure protocol for NFS transactions.

acregmin=*n*
Hold cached attributes for at least *n* seconds after file modification.
acregmax=*n*
Hold cached attributes for no more than *n* seconds after file modification.
acdirmin=*n*
Hold cached attributes for at least *n* seconds after directory update.
acdirmax=*n*
Hold cached attributes for no more than *n* seconds after directory update.
actimeo=*n*
Set *min* and *max* times for regular files and directories to *n* seconds.
noac Suppress attribute caching.

Regular defaults are:
fg,retry=10000,timeo=7,retrans=3,port=NFS_PORT,hard,
acregmin=3,acregmax=60,acdirmin=30,acdirmax=60

actimeo has no default; it sets **acregmin**, **acregmax**, **acdirmin** and **acdirmax**

Defaults for **rsize** and **wsize** are set internally by the system kernel.

rfs options:

bg | fg If the first attempt fails, retry in the background, or, in the foreground.
retry=*n* The number of times to retry the mount operation.

Defaults are the same as for NFS.

Common options:

ro | rw mount either read-only or read-write
suid | nosuid
setuid execution allowed or disallowed
grpid Create files with BSD semantics for propagation of the group ID. With this option, files inherit the group ID of the directory in which they are created, regardless of the directory's setgid bit.
noauto Do not mount this file system automatically (using '**mount –a**').

freq is the interval (in days) between dumps.

pass is the **fsck**(8) pass in which to check the partition. Filesystems with *pass* 0 are not checked. Filesystems with the pass 1 are checked sequentially. In general, the root filesystem should be checked in pass 1, with others checked in higher (later) passes. For passes higher than 1, multiple filesystems in the same pass are checked simultaneously.

A hash-sign (#) as the first character indicates a comment line which is ignored by routines that read this file. The order of records in **/etc/fstab** is important because **fsck**, **mount**, and **umount** process the file sequentially; an entry for a file system must appear *after* the entry for any file system it is to be mounted on top of.

EXAMPLES

In this example, two partitions on the local disk are **4.2** mounted. Several **/export** directories are loopback mounted to appear in the traditional file system locations on the local system. The **/home/user** directory is hard mounted read-write over the NFS, along with additional swap space in the form of a mounted swap file (see *System and Network Administration* for details on adding swap space):

```
/dev/xy0a / 4.2 rw,noquota 1 1
/dev/xy0b /usr 4.2 rw,noquota 1 1
/export/tmp/localhost /tmp lo rw 0 0
/export/var/localhost /var lo rw 0 0
/export/cluster/sun386.sunos4.0.1 /usr/cluster lo rw 0 0
/export/local/sun386 /usr/local lo rw 0 0
```

```
example:/home/user /home/user nfs rw,hard,fg 0 0
/export/swap/myswap swap swap rw 0 0
```

FILES

/etc/fstab
/etc/mtab

SEE ALSO

swapon(2), **getmntent**(3), **lofs**(4S), **fsck**(8), **mkfile**(8), **mount**(8), **quotacheck**(8), **quotaon**(8), **swapon**(8)

System and Network Administration

NAME

ftpusers – list of users prohibited by FTP

SYNOPSIS

/etc/ftpusers

DESCRIPTION

ftpusers contains a list of users who cannot access this system using the File Transfer Protocol (FTP). **ftpusers** contains one user name per line.

If this file is missing, the list of users is considered to be empty, so that any user may use FTP to access the system if the other criteria for access are met (see **ftpd**(8C)).

SEE ALSO

ftp(1C), **ftpd**(8C)

NAME

gettytab – terminal configuration data base

SYNOPSIS

/etc/gettytab

DESCRIPTION

gettytab is a simplified version of the **termcap**(5) data base used to describe terminal lines. The initial terminal login process **getty**(8) accesses the **gettytab** file each time it starts, allowing simpler reconfiguration of terminal characteristics. Each entry in the data base is used to describe one class of terminals.

There is a default terminal class, **default**, that is used to set global defaults for all other classes. That is, the **default** entry is read, then the entry for the class required is used to override particular settings.

CAPABILITIES

Refer to **termcap**(5) for a description of the file layout. The *Default* column below lists defaults obtained if there is no entry in the table obtained, nor one in the special default table.

Name	*Type*	*Default*	*Description*
ab	bool	false	read a \r first and guess the baud rate from it
ap	bool	false	terminal uses 7 bits, any parity
bd	num	0	backspace delay
bk	str	0377	alternate end of line character (input break)
cb	bool	false	use crt backspace mode
cd	num	0	carriage-return delay
ce	bool	false	use crt erase algorithm
ck	bool	false	use crt kill algorithm
cl	str	NULL	screen clear sequence
co	bool	false	console - add NEWLINE after login prompt
de	num	0	delay before first prompt is printed (seconds)
ds	str	^Y	delayed suspend character
dx	bool	false	set DECCTLQ
ec	bool	false	leave echo OFF
ep	bool	false	terminal uses 7 bits, even parity
er	str	^?	erase character
et	str	^D	end of text (EOF) character
ev	str	NULL	initial environment
f0	num	unused	tty mode flags to write messages
f1	num	unused	tty mode flags to read login name
f2	num	unused	tty mode flags to leave terminal as
fd	num	0	form-feed (vertical motion) delay
fl	str	^O	output flush character
hc	bool	false	do NOT hangup line on last close
he	str	NULL	hostname editing string
hn	str	hostname	hostname
ht	bool	false	terminal has real tabs
ig	bool	false	ignore garbage characters in login name
im	str	NULL	initial (banner) message
in	str	^C	interrupt character
is	num	unused	input speed
kl	str	^U	kill character
lc	bool	false	terminal has lower case
lm	str	login:	login prompt
ln	str	^V	"literal next" character
lo	str	/usr/bin/login	program to exec when name obtained

ms	str	NULL	list of terminal modes to set or clear
m0	str	NULL	set modes that apply at the same time as those set by f0
m1	str	NULL	set modes that apply at the same time as those set by f1
m2	str	NULL	set modes that apply at the same time as those set by f2
nd	num	0	NEWLINE (LINEFEED) delay
nl	bool	false	terminal has (or might have) a NEWLINE character
nx	str	default	next table (for auto speed selection)
op	bool	false	terminal uses 7 bits, odd parity
os	num	unused	output speed
p8	bool	false	terminal uses 8 bits, no parity
pc	str		pad character
pe	bool	false	use printer (hard copy) erase algorithm
pf	num	0	delay between first prompt and following flush (seconds)
ps	bool	false	line connected to a MICOM port selector
qu	str	^	quit character
rp	str	^R	line retype character
rw	bool	false	do NOT use RAW for input, use CBREAK
sp	num	0	line speed (input and output)
su	str	^Z	suspend character
tc	str	none	table continuation
td	num	0	tab delay
to	num	0	timeout (seconds)
tt	str	NULL	terminal type (for environment)
ub	bool	false	do unbuffered output (of prompts etc)
uc	bool	false	terminal is known upper case only
we	str	^W	word erase character
xc	bool	false	do NOT echo control chars as ^X
xf	str	^S	XOFF (stop output) character
xn	str	^Q	XON (start output) character

If no line speed is specified, speed will not be altered from that which prevails when **getty** is entered. Specifying an input or output speed overrides line speed for stated direction only. If **ab** is specified, **getty** will initially read a character from the tty, assumed to be a carriage return, and will attempt to figure out the baud rate based on what the character appears as. It will then look for a table entry for that baud rate; if the line appears to be a 300 baud line, it will look for an entry **300-baud**, if it appears to be a 1200 baud line, it will look for an entry **1200-baud**, etc..

Terminal modes to be used for the output of the message, for input of the login name, and to leave the terminal set as upon completion, are derived from the Boolean flags specified. If the derivation should prove inadequate, any (or all) of these three may be overridden with one of the **f0**, **f1**, or **f2** numeric specifications, which can be used to specify (usually in octal, with a leading '0') the exact values of the flags. Local (new tty) flags are set in the top 16 bits of this (32 bit) value.

The **ms** field can be used to specify modes to be set and cleared. These modes are specified as **stty**(1V) modes; any mode supported by **stty** may be specified, except for the baud rate which must be specified with the **br** field. This permits modes not supported by the older terminal interface described in **ttcompat**(4M) to be set or cleared. Thus, to set the terminal port to which the printer is attached to even parity, TAB expansion, no NEWLINE to RETURN/LINEFEED translation, and RTS/CTS flow control enabled, do:

:ms=evenp,-tabs,nl,crtscts:

The **m0**, **m1**, and **m2** fields can be used to set modes which only apply concurrently with those set by **f0**, **f1**, and **f2**, respectively. The modes specified by **ms**, **m0**, **m1**, and **m2** are applied *after* the modes specified by other existing capabilities.

Should **getty** receive a null character (presumed to indicate a line break) it will restart using the table indicated by the **nx** entry. If there is none, it will re-use its original table.

Delays are specified in milliseconds, the nearest possible delay available in the tty driver will be used. Should greater certainty be desired, delays with values 0, 1, 2, and 3 are interpreted as choosing that particular delay algorithm from the driver.

The **cl** screen clear string may be preceded by a (decimal) number of milliseconds of delay required (as with **termcap**(5)). This delay is simulated by repeated use of the pad character **pc**.

The initial message, and login message, **im** and **lm** may include the character sequence **%h** or **%t** to obtain the hostname or tty name respectively. (**%%** obtains a single '**%**' character.) The hostname is normally obtained from the system, but may be set by the **hn** table entry. In either case it may be edited with **he**. The **he** string is a sequence of characters, each character that is neither '**@**' nor '**#**' is copied into the final hostname. A '**@**' in the **he** string, copies one character from the real hostname to the final hostname. A '**#**' in the **he** string, skips the next character of the real hostname. Surplus '**@**' and '**#**' characters are ignored.

When **getty** execs the login process, given in the **lo** string (usually **/usr/bin/login**), it will have set the environment to include the terminal type, as indicated by the **tt** string (if it exists). The **ev** string, can be used to enter additional data into the environment. It is a list of comma separated strings, each of which will presumably be of the form *name=value*.

If a non-zero timeout is specified, with **to**, then **getty** will exit within the indicated number of seconds, either having received a login name and passed control to *login*, or having received an alarm signal, and exited. This may be useful to hangup dial in lines.

Output from **getty** is even parity unless **op** or **p8** is specified. **op** may be specified with **ap** to allow any parity on input, but generate odd parity output. Note: this only applies while **getty** is being run, terminal driver limitations prevent a more complete implementation. **getty** does not check parity of input characters in RAW mode.

FILES

/etc/gettytab

SEE ALSO

termcap(5), **getty**(8)

NAME

group – group file

SYNOPSIS

/etc/group

DESCRIPTION

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 is the name of the group.

gid is the group's numerical ID within the system; it must be unique.

user-list is a comma-separated list of users allowed in the group.

If the password field is empty, no password is demanded. The **group** file is an ASCII file. Because of the encrypted passwords, the **group** file can and does have general read permission, and can be used as a mapping of numerical group IDs to group names.

A group entry beginning with a '+' (plus sign), means to incorporate an entry or entries from the Network Information Service (NIS) A '+' on a line by itself means to insert the entire contents of the NIS group file at that point in the file. An entry of the form: '+*groupname*' means to insert the entry (if any) for **groupname**. If a '+' entry has a non-empty *password* or *user-list* field, the contents of that field override the corresponding field from the NIS service. The *gid* field cannot be overridden in this way.

An entry of the form: –*groupname* indicates that the group is disallowed. All subsequent entries for the indicated *groupname*, whether originating from the NIS service, or the local **group** file, are 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**(8) to check the **/etc/group** database from time to time.

Sun386i systems uses the following group IDs as program privileges:

operator	5	Privilege to do backup as root.
accounts	11	Privilege to update user accounts.
networks	12	Privilege to change network configuration.
devices	13	Privilege to modify printer, terminal, or modem configurations.

On all Sun systems, SunOS uses group ID 0 as privilege to run **su**(1V).

EXAMPLE

Here is a sample group file when the **group.adjunct** file does not exist:

```
primary:q.mJzTnu8icF.:10:fred,mary
+myproject:::bill,steve
+:
```

Here is a sample group file when the **group.adjunct** file does exist:

```
primary:#$primary:10:fred,mary
+myproject:::bill,steve
+:
```

If these entries appear at the end of a group file, then the group *primary* will have members **fred** and **mary**, and a group ID of **10**. The group *myproject* will have members **bill** and **steve**, and the password and group ID of the NIS entry for the group **myproject**. All groups listed in the NIS service are pulled in and placed after the entry for **myproject**.

FILES

/etc/group

SEE ALSO

passwd(1), **su**(1V), **getgroups**(2V), **crypt**(3), **initgroups**(3), **group.adjunct**(5), **passwd**(5), **grpck**(8V)

NOTES

SunOS releases prior to SunOS 4.0, permitted a user to belong to no more then eight groups at a time. A user who belongs to more than eight groups may have trouble using the RPC service (and therefore NFS) to communicate with machines running older releases. In such cases, RPC complains of an "Authentication Error".

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.

BUGS

The **passwd**(1) command will not change group passwords.

NAME

group.adjunct – group security data file

SYNOPSIS

/etc/security/group.adjunct

DESCRIPTION

The **group.adjunct** file contains the following information for each group:

groupname:*password*

groupname The group's name in the system; it must be unique.

password The encrypted password, formerly field two of the **/etc/group** file.

The **group.adjunct** file is in ASCII. Fields are separated by a colon, and each group is separated from the next by a NEWLINE.

A **group.adjunct** file can have a line beginning with a '+' (plus sign), which means to incorporate entries from the Network Information Service (NIS). There are two styles of '+' entries: all by itself, '+' means to insert the entire contents of the **group.adjunct** NIS file at that point; *+name* means to insert the entry (if any) for *name* from the NIS service at that point. If a '+' entry has a non-null password, the contents of that field will override what is contained in the NIS service.

FILES

/etc/group

SEE ALSO

crypt(3), **getgraent**(3), **getgrent**(3V), **group**(5)

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.

NAME

help – help file format

SYNOPSIS

/usr/lib/help/*

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

Each SunView application using the **help** feature has a simple ASCII file in **/usr/lib/help** with the name *application-name*.**info**.

This file contains the text of help messages for each SunView object within that program. Each help message is separated in the file by a line beginning with a colon and identified by a keyword which matches the HELP_DATA attribute of the SunView object.

The first character of each line in the file may be:

#	comment line
:	keyword line
any other	1-32 help text lines

If the line is a keyword line, it has the following structure—

:keyword[s]:datastring [pagenumber]NEWLINE

keyword is a 1-65 character keyword
--any displayable characters may be used
--several keywords may be present
--keywords are separated by 1-or-more blanks

datastring is 1-256 ASCII bytes, and describes the path of the data files for help_viewer, relative to /usr/lib/help.

pagenumber is an optional page number within the help_viewer data file.

The help text which follows the :keyword line will be displayed in an Alert Box when help is requested for one of the keywords by pressing the help key.

The datastring will be sent (by RPC) to the **help_viewer** procedure when the user selects the More Help box in the Alert Box window.

EXAMPLE

Here is part of a typical help file, called **mailtool.info**.

```
:abort
  Abort button

o Quits the Mail application (click
  left on button). Tentative message
  deletions do not become permanent.

o Provides a menu of Abort options
  (click right on button).
```

:cancel:mailtool/Writing_and_Sending_Mail 1
Cancel button

o Closes the message composition window without sending message (click left on button).

o Provides a menu of Cancel options (click right on button).

Pressing the help key while in the cancel or abort buttons triggers the display of the corresponding text. The words *cancel* and *abort* in this file are the keywords. In the case of abort, there is no More Help available. For cancel, More Help is available and it is stored in the first page of the **Writing_and_Sending_Mail** file in the mailtool directory.

FILES

/usr/lib/help/* files for the pop-up help facility

SEE ALSO

help_viewer(1), **help_viewer**(5)

Sun386i Developer's Guide

NAME

help_viewer – help viewer file format

SYNOPSIS

/usr/lib/help/*/*

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*.x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

The **help_viewer** reads files of various types. The Top Level list of applications documented is **/usr/lib/help/Top_Level**. The Master Index shown at the top level is **/usr/lib/help/Master_Index**. These files are FrameMaker files. To add or remove a heading from this list, use FrameMaker (1.1 or later).

Each directory within **/usr/lib/help** that corresponds to a SunView application name contains detailed information about that application. These are also FrameMaker files. The ***.rf** files are rasterfiles, of standard image format created by FrameMaker. These are the pictures that are interleaved into the text.

The **Frame/** subdirectory of **/usr/lib/help** contains topic, contents, and index templates which can be used to create new Help Viewer handbooks. The **Interleaf/** subdirectory contains Interleaf templates, fonts, and initialization files.

FILES

/usr/lib/help/*/*

SEE ALSO

help(5), **help_viewer**(1)

NAME

hosts – host name data base

SYNOPSIS

/etc/hosts

DESCRIPTION

The **hosts** file contains information regarding the known hosts on the TCP/IP. For each host a single line should be present with the following information:

Internet-address official-host-name 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. This file is normally created from the official host data base 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 hosts.

Network addresses are specified in the conventional '.' notation using the **inet_addr** ( ) routine from the Internet address manipulation library, **inet**(3N). Host names may contain any printable character other than an upper case character, a field delimiter, NEWLINE, or comment character.

EXAMPLE

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

192.9.1.20 gaia # John Smith

FILES

/etc/hosts

SEE ALSO

gethostent(3N), **inet**(3N)

NAME

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

DESCRIPTION

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

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

The remote authentication procedure determines whether a particular remote user from a particular remote host should be allowed to access the local system as a (possibly different) particular 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 as whom access is being attempted. Entries in these files can be of two forms. *Positive* entries explicitly *allow* access, while *negative* entries explicitly *deny* access. The authentication succeeds as soon as a matching positive entry is found. The procedure fails when a matching negative entry is found, or if no matching entries are found in either file. The order of entries, therefore, can be important: If the files contain both matching positive and negative entries, the entry that appears first will prevail. The **rsh**(1C) and **rcp**(1C) programs fail if the remote authentication procedure fails. The **rlogin** program will fall back to the standard password-based login procedure if the remote authentication fails.

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

hostname [*username*]

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

Positive Entries

If the form:

hostname

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

If the line is in the form:

hostname username

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

Netgroups(5) can be used in either the *hostname* or *username* fields to match a number of hosts or users in one entry. The form:

+@netgroup

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

hostname +@netgroup

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

+@netgroup1 +@netgroup2

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

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

+

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

+ *username*

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

hostname +

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

Negative Entries

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

-*hostname*

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

–@*netgroup*

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

hostname –username

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

+ –@*netgroup*

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

FILES

/etc/hosts.equiv
~/.rhosts

NOTES

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

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

As a security feature, the *.rhosts* file must be owned by the user as whom access is being attempted.

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

SEE ALSO

rlogin(1C), **rsh**(1C), **rcp**(1C), **rcmd**(3N), **hosts**(5), **netgroup**(5), **passwd**(5)

NAME

indent.pro – default options for indent

DESCRIPTION

The **.indent.pro** file in either the current or home directory contains default command line options for the **indent**(1) program. It is a text file that contains space-separated command line options. For a description of these options, see **indent**(1).

Explicit command line options override options taken from **.indent.pro**.

Here is a sample **.indent.pro** file:

```
-bap -nbad -nbbb -bc -br -cdb -nce
-fc1 -ip -lp -npcs -psl -sc -nsob -cli0
-di12 -l79 -i4  -d0 -c33
```

FILES

./.indent.pro
~/.indent.pro

SEE ALSO

indent(1)

NAME

inetd.conf – Internet servers database

DESCRIPTION

The **inetd.conf** file contains the list of servers that **inetd**(8C) invokes when it receives an Internet request over a socket. Each server entry is composed of a single line of the form:

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

Fields can be separated by either spaces or TAB characters. A '#' (pound-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 is the name of a valid service listed in the file **/etc/services**. For RPC services, the value of the *service-name* field consists of the RPC service name, followed by a slash and either a version number or a range of version numbers (for example, **mountd/1**).

socket-type can be one of:

stream	for a stream socket,
dgram	for a datagram socket,
raw	for a raw socket,
rdm	for a "reliably delivered message" socket, or
seqpacket	for a sequenced packet socket.

protocol must be a recognized protocol listed in the file **/etc/protocols**. For RPC services, the field consists of the string "rpc" followed by a slash and the name of the protocol (for example, **rpc/udp** for an RPC service using the UDP protocol as a transport mechanism).

wait-status is **nowait** for all but "single-threaded" datagram servers — servers which do not release the socket until a timeout occurs (such as **comsat**(8C) and **talkd**(8C)). These must have the status **wait**. Although **tftpd**(8C) establishes separate "pseudo-connections", its forking behavior can lead to a race condition unless it is also given the status **wait**.

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

server-program is 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**'.

FILES

/etc/inetd.conf
/etc/services
/etc/protocols

SEE ALSO

services(5), **comsat**(8C), **inetd**(8C), **talkd**(8C), **tftpd**(8C)

BUGS

inetd dumps core when the **inetd.conf** file contains blank lines.

NAME

internat – key mapping table for internationalization

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

This file format is used for the file specified by the –f option of **old-setkeys**(1).

The file has three columns. First column is keytable identifier, one of: BASE, CTRL, SHIFT, CAPS, UP, BASE_ISO, SHIFT_ISO or ALTG. The second column is a decimal keystation number. The third column is hexadecimal keytable entry value. The file must end with line of "END, 0, 0". As usual, comment lines start with #.

EXAMPLES

This is the file for mapping keys to Canadian standards:

```
# /usr/lib/.setkeys: Key remapping, used by "setkeys remap"
#
# First column is keytable identifier:
#                BASE, CTRL, SHIFT, CAPS, UP, BASE_ISO, SHIFT_ISO or ALTG
# Second column is decimal keystation number
# Third column is hexadecimal keytable entry value
# File must end with line of "END, 0, 0"
# Comment lines must start with #
#
#
# --- Keymaps for Canadian keyboard ---
# > Define Alt Graph key (SHIFTKEYS+ALTGRAPH=86)
BASE 119 86
CTRL 119 86
SHIFT 119 86
CAPS 119 86
UP 119 86
# > Define Caps key (SHIFTKEYS+CAPSLOCK=80)
BASE 13 80
CTRL 13 80
SHIFT 13 80
CAPS 13 80
# > Define Floating Accent keys
#               FA_UMLAUT = A9
#               FA_CFLEX = AA
#               FA_TILDE = AB
#               FA_CEDILLA = AC
#               FA_ACUTE = AD
#               FA_GRAVE = AE
BASE 64 AA
SHIFT 64 A9
CAPS 64 A9
BASE 65 AC
SHIFT 65 AB
CAPS 65 AB
BASE 87 AE
SHIFT 87 AD
CAPS 87 AD
# > Define ASCII values
```

```
BASE 88 5B
SHIFT 88 7B
CAPS 88 7B
BASE 15 5D
SHIFT 15 7D
CAPS 15 7D
SHIFT 31 22
SHIFT 32 2F
SHIFT 35 3F
SHIFT 107 27
CAPS 107 27
SHIFT 108 60
CAPS 108 60
BASE 124 3C
SHIFT 124 3E
CAPS 124 3E
# > Define ISO values
BASE_ISO 109 E9
SHIFT_ISO 109 C9
# > Define Alternate Graph ISO values
ALTG 88 AB
ALTG 15 BB
ALTG 30 B1
ALTG 31 B2
ALTG 32 B3
ALTG 33 A2
ALTG 34 A4
ALTG 35 5E
ALTG 36 40
ALTG 37 A3
ALTG 38 5C
ALTG 40 AC
ALTG 41 23
ALTG 63 B6
ALTG 64 BC
ALTG 65 BD
ALTG 42 BE
ALTG 106 B5
ALTG 105 BA
# > End of file
END 0 0
```

SEE ALSO

old-setkeys(1)

The *Sun386i Developer's Guide* for keystation number diagrams.

NAME

ipalloc.netrange – range of addresses to allocate

SYNOPSIS

/etc/ipalloc.netrange

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

This file, if it exists on the Network Information Service (NIS) master of the **hosts.byaddr** map, specifies the ranges of IP addresses that can be allocated by the **ipallocd**(8C) daemon. This allows multiple address assignment authorities, probably in multiple administrative domains, to coexist on the same IP network by preallocating ranges of addresses. If this file does not exist, the daemon assumes that all addresses not listed in the **hosts** map may be freely allocated.

This file can contain blank lines. Comments begin with a '#' character and extend to the end of the current line. Ranges of free addresses are specified on one line per network or subnetwork.

The first token on the line is the IP address, in four part "dot" notation as also used in the **hosts** file, of the network or subnetwork described. It is separated from the second token by white space. The second token is a comma-separated list of local host number ranges on that network. These ranges take two forms: a single number specifies just that local host number, and two numbers separated by a dash specify all local host numbers starting at the first number and ending at the second. In the case of a subnet, host numbers not in that subnet are excluded.

For example, the following file would specify that a subset of the addresses on the class C network 192.9.200.0 may be allocated, and only some of the addresses on two particular subnets of the class B network 128.255.0.0 may be allocated. In any case, only non-broadcast addresses not listed in the **hosts** map are subject to allocation:

```
# We have three network cables administered using automatic # IP address allocation.

192.9.200.0     50-100,200-254     128.255.210.0     3,5,7,9,100-110
128.255.211.0   1-254
```

SEE ALSO

hosts(5), **netmasks**(5), **ipallocd**(8C)

BUGS

There is a silent limit of twenty ranges per network.

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.

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 **kb**(4M), and generated by **dumpkeys** (see **loadkeys**(1)) from those translation tables.

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

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

key *number list_of_entries*

or

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

or

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

or a blank line, which is ignored.

key *number list_of_entries*

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

tablename code

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

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

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

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

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

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

- One of the following expressions:

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

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

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

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

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

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

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

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

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

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

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

compose the key is to be the "Compose" key

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

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

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

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

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

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

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

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

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

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

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

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

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

fa_umlaut	the key is to be the umlaut "floating accent" key
nonl	this is used only in the Num Lock table; the key is not to be affected by the state of Num Lock
pad0	the key is to be the "0" key on the numeric keypad
pad1	the key is to be the "1" key on the numeric keypad
pad2	the key is to be the "2" key on the numeric keypad
pad3	the key is to be the "3" key on the numeric keypad
pad4	the key is to be the "4" key on the numeric keypad
pad5	the key is to be the "5" key on the numeric keypad
pad6	the key is to be the "6" key on the numeric keypad
pad7	the key is to be the "7" key on the numeric keypad
pad8	the key is to be the "8" key on the numeric keypad
pad9	the key is to be the "9" key on the numeric keypad
paddot	the key is to be the "." key on the numeric keypad
padenter	the key is to be the "Enter" key on the numeric keypad
padplus	the key is to be the "+" key on the numeric keypad
padminus	the key is to be the "-" key on the numeric keypad
padstar	the key is to be the "*" key on the numeric keypad
padslash	the key is to be the "/" key on the numeric keypad
padequal	the key is to be the "=" key on the numeric keypad
padsep	the key is to be the "," (separator) key on the numeric keypad
lf(*n*)	the key is to be the left-hand function key *n*
rf(*n*)	the key is to be the right-hand function key *n*
tf(*n*)	the key is to be the top function key *n*
bf(*n*)	the key is to be the "bottom" function key *n*
nop	the key is to do nothing
error	this code indicates an internal error; to be used only for keystation 126, and must be used there
idle	this code indicates that the keyboard is idle (that is, has no keys down); to be used only for all entries other than the **numl** and **up** table entries for keystation 127, and must be used there
oops	this key exists, but its action is not defined; it has the same effect as **nop**
reset	this code indicates that the keyboard has just been reset; to be used only for the **up** table entry for keystation 127, and must be used there

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

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

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

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

EXAMPLES

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

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

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

```
swap 43 with 66
```

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

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

```
key 119 all nop
```

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

```
key 0   all hole
key 1   all buckybits+systembit up buckybits+systembit
key 2   all hole
key 3   all lf(2)
key 4   all hole
key 5   all tf(1)
key 6   all tf(2)
key 7   all tf(10)
key 8   all tf(3)
key 9   all tf(11)
key 10  all tf(4)
key 11  all tf(12)
key 12  all tf(5)
key 13  all shiftkeys+altgraph up shiftkeys+altgraph
key 14  all tf(6)
key 15  all hole
key 16  all tf(7)
key 17  all tf(8)
key 18  all tf(9)
key 19  all shiftkeys+alt up shiftkeys+alt
key 20  all hole
key 21  all rf(1)
key 22  all rf(2)
key 23  all rf(3)
key 24  all hole
key 25  all lf(3)
key 26  all lf(4)
key 27  all hole
key 28  all hole
key 29  all ^[
key 30  base 1 shift ! caps 1 ctrl 1 altg nop
key 31  base 2 shift @ caps 2 ctrl ^@ altg nop
key 32  base 3 shift # caps 3 ctrl 3 altg nop
key 33  base 4 shift $ caps 4 ctrl 4 altg nop
key 34  base 5 shift % caps 5 ctrl 5 altg nop
key 35  base 6 shift ^ caps 6 ctrl ^^ altg nop
key 36  base 7 shift & caps 7 ctrl 7 altg nop
```

```
key 37   base 8 shift * caps 8 ctrl 8 altg nop
key 38   base 9 shift ( caps 9 ctrl 9 altg nop
key 39   base 0 shift ) caps 0 ctrl 0 altg nop
key 40   base - shift _ caps - ctrl ^_ altg nop
key 41   base = shift + caps = ctrl = altg nop
key 42   base ` shift ~ caps ` ctrl ^^ altg nop
key 43   all '\b'
key 44   all hole
key 45   all rf(4) numl padequal
key 46   all rf(5) numl padslash
key 47   all rf(6) numl padstar
key 48   all bf(13)
key 49   all lf(5)
key 50   all bf(10) numl padequal
key 51   all lf(6)
key 52   all hole
key 53   all '\t'
key 54   base q shift Q caps Q ctrl ^Q altg nop
key 55   base w shift W caps W ctrl ^W altg nop
key 56   base e shift E caps E ctrl ^E altg nop
key 57   base r shift R caps R ctrl ^R altg nop
key 58   base t shift T caps T ctrl ^T altg nop
key 59   base y shift Y caps Y ctrl ^Y altg nop
key 60   base u shift U caps U ctrl ^U altg nop
key 61   base i shift I caps I ctrl '\t' altg nop
key 62   base o shift O caps O ctrl ^O altg nop
key 63   base p shift P caps P ctrl ^P altg nop
key 64   base [ shift { caps [ ctrl ^[ altg nop
key 65   base ] shift } caps ] ctrl ^] altg nop
key 66   all '\177'
key 67   all compose
key 68   all rf(7) numl pad7
key 69   all rf(8) numl pad8
key 70   all rf(9) numl pad9
key 71   all bf(15) numl padminus
key 72   all lf(7)
key 73   all lf(8)
key 74   all hole
key 75   all hole
key 76   all shiftkeys+leftctrl up shiftkeys+leftctrl
key 77   base a shift A caps A ctrl ^A altg nop
key 78   base s shift S caps S ctrl ^S altg nop
key 79   base d shift D caps D ctrl ^D altg nop
key 80   base f shift F caps F ctrl ^F altg nop
key 81   base g shift G caps G ctrl ^G altg nop
key 82   base h shift H caps H ctrl '\b' altg nop
key 83   base j shift J caps J ctrl '\n' altg nop
key 84   base k shift K caps K ctrl '\v' altg nop
key 85   base l shift L caps L ctrl ^L altg nop
key 86   base ; shift : caps ; ctrl ; altg nop
key 87   base '\'' shift '"' caps '\'' ctrl '\'' altg nop
key 88   base '\\' shift | caps '\\' ctrl ^\ altg nop
key 89   all '\r'
```

```
key 90   all bf(11) numl padenter
key 91   all rf(10) numl pad4
key 92   all rf(11) numl pad5
key 93   all rf(12) numl pad6
key 94   all bf(8) numl pad0
key 95   all lf(9)
key 96   all hole
key 97   all lf(10)
key 98   all shiftkeys+numlock
key 99   all shiftkeys+leftshift up shiftkeys+leftshift
key 100  base z shift Z caps Z ctrl ^Z altg nop
key 101  base x shift X caps X ctrl ^X altg nop
key 102  base c shift C caps C ctrl ^C altg nop
key 103  base v shift V caps V ctrl ^V altg nop
key 104  base b shift B caps B ctrl ^B altg nop
key 105  base n shift N caps N ctrl ^N altg nop
key 106  base m shift M caps M ctrl '\r' altg nop
key 107  base , shift < caps , ctrl , altg nop
key 108  base . shift > caps . ctrl . altg nop
key 109  base / shift ? caps / ctrl ^_ altg nop
key 110  all shiftkeys+rightshift up shiftkeys+rightshift
key 111  all '\n'
key 112  all rf(13) numl pad1
key 113  all rf(14) numl pad2
key 114  all rf(15) numl pad3
key 115  all hole
key 116  all hole
key 117  all hole
key 118  all lf(16)
key 119  all shiftkeys+capslock
key 120  all buckybits+metabit up buckybits+metabit
key 121  base ' ' shift ' ' caps ' ' ctrl ^@ altg ' '
key 122  all buckybits+metabit up buckybits+metabit
key 123  all hole
key 124  all hole
key 125  all bf(14) numl padplus
key 126  all error numl error up hole
key 127  all idle numl idle up reset
```

SEE ALSO

loadkeys(1), **kb**(4M)

NAME

link – link editor interfaces

SYNOPSIS

#include <link.h>

DESCRIPTION

Dynamically linked executables created by **ld**(1) contain data structures used by the dynamic link editor to finish link-editing the program during program execution. These data structures are described with a **link_dynamic** structure, as defined in the **link.h** file. **ld** always identifies the location of this structure in the executable file with the symbol **__DYNAMIC**. This symbol is **ld**-defined and if referenced in an executable that does not require dynamic linking will have the value zero.

The program stub linked with "main" programs by compiler drivers such as **cc**(1V) (called **crt0**) tests the definition of **__DYNAMIC** to determine whether or not the dynamic link editor should be invoked. Programs supplying a substitute for **crt0** must either duplicate this functionality or else require that the programs with which they are linked be linked *statically*. Otherwise, such replacement **crt0**'s must open and map in the executable **/usr/lib/ld.so** using **mmap**(2). Care should be taken to ensure that the expected mapping relationship between the "text" and "data" segments of the executable is maintained in the same manner that the **execve**(2V) system call does. The first location following the **a.out** header of this executable is the entry point to a function that begins the dynamic link-editing process. This function must be called and supplied with two arguments. The first argument is an integer representing the revision level of the argument list, and should have the value "1". The second should be a pointer to an argument list structure of the form:

```
struct {
        int         crt_ba;          /* base address of ld.so */
        int         crt_dzfd;        /* open fd to /dev/zero */
        int         crt_ldfd;        /* open fd to ld.so */
        struct      link_dynamic *crt_dp; /* pointer to program's __DYNAMIC */
        char        **crt_ep;        /* environment strings */
        caddr_t     crt_bp;          /* debugger hook */
}
```

The members of the structure are:

crt_ba The address at which **/usr/lib/ld.so** has been mapped.

crt_dzfd An open file descriptor for **/dev/zero**. **ld.so** will close this file descriptor before returning.

crt_ldfd The file descriptor used to map **/usr/lib/ld.so**. **ld.so** will close this file descriptor before returning.

crt_dp A pointer to the label **__DYNAMIC** in the executable which is calling **ld.so**.

crt_ep A pointer to the environment strings provided to the program.

crt_bp A location in the executable which contains an instruction that will be executed after the call to **ld.so** returns. This location is used as a breakpoint in programs that are being executed under the control of a debugger such as **adb**(1).

SEE ALSO

ld(1), **mmap**(2), **a.out**(5)

BUGS

These interfaces are under development and are subject to rapid change.

NAME

locale – locale database

SYNOPSIS

/usr/share/lib/locale/*category/locale*

/etc/locale/*category/locale*

DESCRIPTION

The *category* directory contains information relating to one category of the complete list of categories that comprise a full locale for all systems sharing this directory. *locale* is either a file or a directory that contains information relating to the relevant category indicated by its parent directory *category*. *locale* is the name that is given to describe the style of operation required by an application in a particular language, territory or code-set.

At runtime these directories will be accessed if the application has made a valid call to:

setlocale(*category, locale***)**

where *category* can be any one of the following settings:

LC_COLLATE Collation order. Affects the behavior of regular expressions and the string functions defined in **strcoll**(3).

LC_CTYPE Character classification and case conversion. Affects the behavior of regular expressions and the character handling functions defined in **toascii**(3), and **ctime**(3V).

LC_MONETARY Monetary formatting. Affects the behavior of functions that handle monetary values.

LC_NUMERIC Numeric delimiters. Affects the radix character of the formatted input/output functions defined in **printf**(3V) and **scanf**(3V), and the conversion functions defined in **strtod**(3).

LC_TIME Date and time formats. Affects the behavior of the time functions defined in **ctime**(3V).

LC_MESSAGES Message presentation style. Affects the behavior of the string access functions defined in **catgets**(3C) and **gettext**(3).

NLSPATH Contains a sequence of pseudo-pathnames which **catopen**(3C) uses when attempting to locate message catalogs. Each pseudo-pathname contains a name template consisting of an optional path-prefix, one or more substitution fields, a filename and an optional filename suffix.

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

%N The value of the *name* parameter passed to **catopen**(3C).

%L The value of the **LANG** environment variable.

%% A single % character.

A null string is sustituted if the specified value is not defined. Pathnames defined in **NLSPATH** are separated by colons (:). A leading or two adjacent colons indicate the current directory. For example:

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

Indicates to **catopen**(3C) that it should look for the requested message catalog in *name, name.cat* and */nlslib/*$LANG/*name.cat*. The **LC_ALL** and **LANG** environment variables do not commute to real directories or files but instead relate to a locale that is a assumed to be valid for all of the above categories.

SEE ALSO

catgets(3C), **catopen**(3C), **ctime**(3V), **gettext**(3), **printf**(3V), **scanf**(3V), **setlocale**(3V), **strcoll**(3) **strtod**(3), **toascii**(3V)

NAME

magic – file command's magic number file

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 file **/etc/magic** specifies what magic numbers are to be tested for, what message to print if a particular magic number is found, and additional information to extract from the file.

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

offset *type* *value* *message*

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

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

byte A one-byte value.

short A two-byte value.

long A four-byte value.

string A string of bytes.

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

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

Numeric values may be preceded by a character indicating the operation to be performed. It may be '=', to specify that the value from the file must equal the specified value, '<', to specify that the value from the file must be less than the specified value, '>', to specify that the value from the file must be greater than the specified value, '&', to specify that all the bits in the specified value must be set in the value from the file, '^', to specify that at least one of the bits in the specified value must not be set in the value from the file, or x to specify that any value will match. If the character is omitted, it is assumed to be '='.

For string values, the byte string from the file must match the specified byte string. The byte string from the file which is matched is the same length as the specified byte string.

message The message to be printed if the comparison succeeds. If the string contains a **printf**(3V) 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), **printf**(3V)

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

mtab – mounted file system table

SYNOPSIS

/etc/mtab

#include <mntent.h>

DESCRIPTION

mtab resides in the **/etc** directory, and contains a table of filesystems currently mounted by the **mount**(8) command. **umount** removes entries from this file.

The file contains a line of information for each mounted filesystem, structurally identical to the contents of **/etc/fstab**, described in **fstab**(5). There are a number of lines of the form:

fsname dir type opts freq passno

for example:

/dev/xy0a / 4.2 rw,noquota 1 2

The file is accessed by programs using **getmntent**(3), and by the system administrator using a text editor.

FILES

/etc/mtab
/etc/fstab

SEE ALSO

getmntent(3), **fstab**(5), **mount**(8)

NAME

netgroup – list of network groups

DESCRIPTION

netgroup defines network wide groups, used for permission checking when doing remote mounts, remote logins, and remote shells. For remote mounts, the information in **netgroup** is used to classify machines; for remote logins and remote shells, it is used to classify users. Each line of the **netgroup** file defines a group and has the format

groupname list-of-members

where members is either another group name, or a triple:

(*hostname*, *username*, *domainname*)

Any of these three fields can be empty, in which case it signifies a wild card. Thus

universal (,,)

defines a group to which everyone belongs.

The *domainname* field must either be the local domain name or empty for the netgroup entry to be used. This field does *not* limit the netgroup or provide security. The *domainname* field refers to the domain in which the triple is valid, not the domain containing the trusted host.

A gateway machine should be listed under all possible hostnames by which it may be recognized:

wan (gateway,,) (gateway-ebb,,)

Field names that begin with something other than a letter, digit or underscore (such as '–') work in precisely the opposite fashion. For example, consider the following entries:

justmachines (analytica,–,sun)
justpeople (–,babbage,sun)

The machine **analytica** belongs to the group **justmachines** in the domain **sun**, but no users belong to it. Similarly, the user **babbage** belongs to the group **justpeople** in the domain **sun**, but no machines belong to it.

SEE ALSO

getnetgrent(3N), **exports**(5), **makedbm**(8), **ypserv**(8)

WARNINGS

The triple, (,,*domain*), allows all users and machines trusted access, and has the same effect as the triple, (,,).

To correctly restrict access to a specific set of members, use the *hostname* and *username* fields of the triple.

NAME

netmasks – network mask data base

DESCRIPTION

The **netmasks** file contains network masks used to implement IP standard subnetting. For each network that is subnetted, a single line should exist in this file with the network number, any number of SPACE or TAB characters, and the network mask to use on that network. Network numbers and masks may be specified in the conventional IP '.' 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 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 running the Network Information Service (NIS), this file on the master is used for the **netmasks.byaddr** map.

FILES

/etc/netmasks

SEE ALSO

ifconfig(8C)

Postel, Jon, and Mogul, Jeff, *Internet Standard Subnetting Procedure*, RFC 950, Network Information Center, SRI International, Menlo Park, Calif., August 1985.

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.

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**(1C). 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**(1V)).

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 as the **ftp macdef** command functions. 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.

EXAMPLE

The command:

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(1V), **ftp**(1C), **ftpd**(8C)

NAME

networks – network name data base

DESCRIPTION

The **networks** file contains information regarding the known networks which comprise the TCP/IP. For each network a single line should be present with the following information:

official-network-name network-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. This file is normally created from the official network data base 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 number may be specified in the conventional '.' notation using the **inet_network ()** routine from the Internet address manipulation library, **inet**(3N). Network names may contain any printable character other than a field delimiter, NEWLINE, or comment character.

FILES

/etc/networks

SEE ALSO

getnetent(3N), **inet**(3N)

BUGS

A name server should be used instead of a static file. A binary indexed file format should be available for fast access.

NAME

orgrc – organizer configuration and initialization file

AVAILABILITY

Sun386i systems only.

DESCRIPTION

organizer(1) is a SunView 1 application for viewing and manipulating files and directories. It saves its parameters in the **.orgrc** file between runs. The user can use this file to configure **organizer**.

The first parameter in the file should always be the version number.

Version = 1.1

Change the version number only when necessary; if **organizer** determines that this version is "old", then it will save this version in **~/.orgrc.old** and try to copy **/usr/lib/Orgrc** into **~/.orgrc**.

The next two parameters assign default names for the system DOS Program and the default text editor.

DOS Program = dos
Text Editor = textedit

The DOS Program parameter should not be changed. However, the user can change the default text editor. For example:

Text Editor = shelltool vi

The Properties section initializes or customizes certain properties. The possible values for each item are listed below. The braces and vertical bars below indicate choices, they are not used in the **.orgrc** file. The **Update Interval** is in seconds.

 # Properties
PROPERTY Display Style = {Name and Icon | Name Only | Name and Info}
PROPERTY Roadmap = {Yes | No}
PROPERTY Show Hidden Files = {Yes | No}
PROPERTY Sort Type = {Name | File Type | Size | Date}
PROPERTY Sort Direction = {Ascending | Descending}
PROPERTY Update Interval = [5-300]

The Color Palette specifies all the color values used by **organizer**'s buttons and icons. These values must be RGB triplets. It is listed below.

Begin Color Palette
Background Color = 255, 255, 255
Directory Name Color = 0, 146, 236
Directory Icon Foreground Color = 114, 45, 0
Directory Icon Background Color = 255, 227, 185
Directory Highlight Name Color = 255, 255, 255
Text Name Color = 0, 166, 143
Text Icon Foreground Color = 0, 0, 0
Text Icon Background Color = 255, 255, 255
Text Highlight Name Color = 255, 255, 255
Executable Name Color = 255, 0, 0
Executable Icon Foreground Color = 157, 162, 187
Executable Icon Background Color = 255, 255, 255
Executable Highlight Name Color = 255, 255, 255
Device Name Color = 113, 117, 135
Device Icon Foreground Color = 0, 0, 0
Device Icon Background Color = 174, 255, 159
Device Highlight Name Color = 255, 255, 255
Button Group1 Color = 255, 220, 187

```
    Button Group2 Color = 201, 211, 232
    Button Group3 Color = 255, 244, 113
    Button Foreground Color = 0, 0, 0
    Button Background Color = 255, 255, 255
    Button Shadow Color = 180, 180, 184
    Button Highlight Color = 0, 0, 0
    Scrollbar Color = 142, 106, 146
  End Color Palette
```

The Color Labels section allows the labelling or "aliasing" of RGB triplets. The right side of a label assignment can contain an RGB triplet, a palette entry, or another label that has already been assigned. Here's an example:

```
  Begin Color Labels
    Black = Text Icon Foreground Color
    White = Background Color
    Orange = 255, 213, 127
    Dark Red = 232, 0, 0
    Steel Blue = 114, 146, 161
    Rasberry (sic) = 202, 140, 156
    Dark Blue = 0, 75, 161
    Light Gray = 223, 223, 223
    Maroon = 182, 84, 106
  End Color Labels
```

The rest of the **.orgrc** file contains user defined file types. The user can specify that certain files be grouped together and treated in a similar fashion. That is, the same icon is used to display all files in a file type, and the same command is used when a file is opened or edited. In the default **.orgrc** (**/usr/lib/Orgrc**) there are ten user defined file types. Here is an example of a user defined file type:

```
  Begin File Type Definition
    Name = *.c
    Background Icon = /usr/include/images/cMask.icon
    Foreground Icon = /usr/include/images/cStencil.icon
    Name Color = Black
    Icon Background Color = Orange
    Icon Foreground Color = Black
    Highlight Name Color = White
    Execute Application = cmdtool vi "$(FILE)"
    Edit Application = cmdtool vi "$(FILE)"
    Print Application = pr -f "$(FILE)" | lpr
  End File Type Definition
```

The right side of the **Name** field can contain any combination of **csh**(1) **Filename Substitution** characters. This field specifies the file type by way of its name. The next six fields together specify an **organizer** icon. This model allows a rich variety of icons. For more information, see the *Sun386i Advanced Skills* manual. The right side of the **Execute Application** entry specifies the command to execute when the user either opens or double clicks on a file of that type. The **Edit Application** and **Print Application** entries specify the command to execute when the user requests that a file of that type be edited or printed.

FILES

~/.orgrc	read at beginning of execution by the Organizer
/usr/lib/Orgrc	default **.orgrc** file

SEE ALSO

organizer(1)

Sun386i User's Guide
Sun386i Advanced Skills

LIMITATIONS

The right side of Color Palette entries must be RGB triplets.

Forward references for Color Labels are not allowed.

BUGS

organizer saves its parameters as it exits; unfortunately, it does not know how to save user's comments in the file. So, comments are blown away.

NAME

passwd – password file

SYNOPSIS

/etc/passwd

DESCRIPTION

The **passwd** file contains basic information about each user's account. This file contains a one-line entry for each authorized user, of the form:

username:*password*:*uid*:*gid*:*gcos-field*:*home-dir*:*login-shell*

where

username is the user's login name. This field contains no uppercase characters, and must not be more than eight characters in length.

password is the user's encrypted password, or a string of the form: *##name* if the encrypted password is in the **/etc/security/passwd.adjunct** file (see **passwd.adjunct**(5)). If this field is empty, **login**(1) does not request a password before logging the user in.

uid is the user's numerical ID for the system, which must be unique. *uid* is generally a value between 0 and 32767.

gid is the numerical ID of the group that the user belongs to. *gid* is generally a value between 0 an 32767.

gcos-field is the user's real name, along with information to pass along in a mail-message heading. It is called the gcos-field for historical reasons. A & 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 **passwd** file can also have lines beginning with a '+' (plus sign) which means to incorporate entries from the Network Information Service (NIS). There are three styles of + entries in this file: by itself, + means to insert the entire contents of the NIS password file at that point; *+name* means to insert the entry (if any) for *name* from the NIS service at that point; *+@netgroup* means to insert the entries for all members of the network group **netgroup** at that point. If a *+name* entry has a non-null *password*, *gcos*, *home-dir*, or *login-shell* field, the value of that field overrides what is contained in the NIS service. The *uid* and *gid* fields cannot be overridden.

The **passwd** file can also have lines beginning with a '–' (minus sign) which means to disallow entries from the NIS service. There are two styles of '–' entries in this file: *–name* means to disallow any subsequent entries (if any) for *name* (in this file or in the NIS service); *–@netgroup* means to disallow any subsequent entries for all members of the network group *netgroup*.

The password file is an ASCII file that resides in the **/etc** directory. Because the encrypted passwords on a secure system are kept in the **passwd.adjunct** file, **/etc/passwd** has general read permission on all systems, and can be used by routines that map numerical user IDs to names.

Appropriate precautions must be taken to lock the **/etc/passwd** file against simultaneous changes if it is to be edited with a text editor; **vipw**(8) does the necessary locking.

EXAMPLE

Here is a sample **passwd** file when **passwd.adjunct** does not exist:

```
root:q.mJzTnu8icF.:0:10:God:/:/bin/csh
fred:6k/7KCFRPNVXg:508:10:% Fredericks:/usr2/fred:/bin/csh
+john:
+@documentation:no-login:
+::::Guest
```

Here is a sample **passwd** file when **passwd.adjunct** does exist:

```
root:##root:0:10:God:/:/bin/csh
fred:##fred:508:10:& Fredericks:/usr2/fred:/bin/csh
+john:
+@documentation:no-login:
+::::Guest
```

In this example, there are specific entries for users **root** and **fred**, to assure that they can log in even when the system is running standalone. The user **john** will have his password entry in the NIS service incorporated without change; anyone in the netgroup **documentation** will have their password field disabled, and anyone else will be able to log in with their usual password, shell, and home directory, but with a gcos-field of **Guest**.

FILES

/etc/passwd
/etc/security/passwd.adjunct

SEE ALSO

login(1), **mail**(1), **passwd**(1), **crypt**(3), **getpwent**(3V), **group**(5), **passwd.adjunct**(5), **adduser**(8), **sendmail**(8), **vipw**(8)

BUGS

mail(1) and **sendmail**(8) use the gcos-field to compose the **From:** line for addressing mail messages, but these programs get confused by nested parentheses when composing replies. This problem can be avoided by using different types of brackets within the gcos-field; for example:

(& Fredricks [Podunk U <EE/CIS>] {818}-555-5555)

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.

NAME

passwd.adjunct – user security data file

SYNOPSIS

/etc/security/passwd.adjunct

DESCRIPTION

The **passwd.adjunct** file contains the following information for each user:

name:*password*:*min-label*:*max-label*:*default-label*:*always-audit-flags*:*never-audit-flags*:

name	The user's login name in the system and it must be unique.
password	The encrypted password.
min-label	The lowest security level at which this user is allowed to login (not used at C2 level).
max-label	The highest security level at which this user is allowed to login (not used at C2 level).
default-label	The security level at which this user will run unless a label is specified at login.
always-audit-flags	Flags specifying events always to be audited for this user's processes; see **audit_control**(5).
never-audit-flags	Flags specifying events never to be audited for this user's processes; see **audit_control**(5).

Field are separated by a colon, and each user from the next by a NEWLINE.

The **passwd.adjunct** file can also have line beginning with a '+' (plus sign), which means to incorporate entries from the Network Information Service (NIS). There are three styles of '+' entries: all by itself, '+' means to insert the entire contents of the NIS **passwd.adjunct** file at that point; *+name* means to insert the entry (if any) for *name* from the NIS service at that point; *+@name* means to insert the entries for all members of the network group *name* at that point. If a '+' entry has a non-null password, it will override what is contained in the NIS service.

EXAMPLE

Here is a sample **/etc/security/passwd.adjunct** file:

```
root:q.mJzTnu8icF.::::::
ignatz:7KsI8CFRPNVXg::b,ap,bp,gp,dp,ic,r,d,l::+dc,+da:-dr:
rex:7HU8UUGRPNVXg:b,ap:b,ap,bp:b,bp::+ad:
+fred:9x.FFUw6xcJBa::::::
+:
```

The user **root** is the super-user, who has no special label constraints nor audit interest. The user **ignatz** may have any label from the lowest to the level **b** and any of a large number of categories. **ignatz** will run at system low unless he specifies otherwise. He is being audited on the system default event classes as well as data creations and access changes, but never for failed data reads. The user **rex** can function only at the level **b** and only in the categories **ap** or **ap** and **bp**. By default, he will run at '**b,bp**'. He is audited with the system defaults, except that successful administrative operations are not audited. The user **fred** will have the labels and audit flags that are specified in the NIS **passwd.adjunct** file. Any other users specified in the NIS service will be able to log in on this system.

The user security data file resides in the **/etc/security** directory. Because it contains encrypted passwords, it does not have general read permission.

FILES

/etc/security/passwd.adjunct
/etc/security

SEE ALSO

login(1), **passwd**(1), **crypt**(3), **getpwaent**(3), **getpwent**(3V), **audit_control**(5), **passwd**(5), **adduser**(8)

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.

NAME

phones – remote host phone number data base

SYNOPSIS

/etc/phones

DESCRIPTION

The file **/etc/phones** contains the system-wide private phone numbers for the **tip**(1C) 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**(5) 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**(1C) will attempt to dial each one in turn, until it establishes a connection.

FILES

/etc/phones

SEE ALSO

tip(1C), **remote**(5)

NAME

plot – graphics interface

DESCRIPTION

Files of this format are produced by routines described in **plot**(3X), and are interpreted for various devices by commands described in **plot**(1G). 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.

Each of the following descriptions begins with the name of the corresponding routine in **plot**(3X).

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**(1G).

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**(1G). 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(1G), **plot**(1G), **plot**(3X)

NAME

pnp.sysnames – file used to allocate system names

SYNOPSIS

/etc/pnp.sysnames

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

The **/etc/pnp.sysnames** file contains system names that may be allocated on demand, typically as part of Automatic System Installation.

The system names should be legal system names, one per line. Legal names are up to 31 characters long, and consist of lowercase alphanumeric characters, dashes, and underscores. The first character must be alphabetic, and the last character should be alphanumeric. Blank lines are allowed in the file, but comments are not.

When a system name needs to be allocated, the first unused system name is taken from **/etc/pnp.sysnames**. If all the system names there are in use, unused names are allocated from the list *system-1*, *system-2*, ...; the default prefix *system* may be changed in the **/var/yp/updaters** makefile. A system name is "used" if there is already a matching entry in the Network Information Service (NIS) *hosts.byname* map, the *ethers.byname* map, or there is a netgroup with that name. Names are allocated to correspond to a given Ethernet address. There is no concept of "transient" name allocation; part of allocating a system name includes updating the *ethers.byname* and *ethers.byaddr* NIS maps to persistently associate the name with that Ethernet address.

One way to allocate a system name is to issue a **ypupdate**(3N) call to update the *ethers.byaddr* map. The key is the Ethernet address (or general IEEE 802.2 48 bit address, used also with FDDI and Token Ring standards) of the system whose name is being allocated. The data is a line formatted according to the format specified in **ethers**(5). A name is allocated if the name passed is '*' (a single asterisk). Updating this NIS map using **ypupdate**(3N) is a privileged operation, and may be performed only by users in the *networks* group (with group ID 12), or boot servers (listed in the *ypservers* NIS map).

FILES

/etc/pnp.sysnames
/usr/etc/yp/upd.systems
/var/yp/updaters

SEE ALSO

ypupdate(3N), **ethers**(5), **group**(5), **hosts**(5), **netgroup**(5), **updaters**(5), **pnpd**(8C)

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.

NAME

policies – network administration policies

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

The **policies** file contains information relevant to domain-wide administration policies. Each line contains two tokens, separated by white space; the first token is the name of an administrative policy, and the second is the value of that policy.

FILES

/etc/policies
/var/yp/*domainname***/policies.{dir,pag}**

SEE ALSO

pnpd(8C), **rarpd**(8C), **logintool**(8)

NAME

printcap – printer capability data base

SYNOPSIS

/etc/printcap

DESCRIPTION

printcap is a simplified version of the **termcap**(5) data base for describing printers. The spooling system accesses the **printcap** file every time it is used, allowing dynamic addition and deletion of printers. Each entry in the data base describes one printer. This data base may not be substituted for, as is possible for **termcap**, because it may allow accounting to be bypassed.

The default printer is normally **lp**, though the environment variable PRINTER may be used to override this. Each spooling utility supports a –P*printer* option to explicitly name a destination printer.

Refer to *System and Network Administration* for a discussion of how to set up the database for a given printer. On Sun386i systems, refer to **snap**(1) for information on setting up printers with the system and network administration program.

Each entry in the **printcap** file describes a printer, and is a line consisting of a number of fields separated by '**:**' characters. The first entry for each printer gives the names which are known for the printer, separated by '|' characters. The first name is conventionally a number. The second name given is the most common abbreviation for the printer, and the last name given should be a long name fully identifying the printer. The second name should contain no blanks; the last name may well contain blanks for readability. Entries may continue onto multiple lines by giving a '\' as the last character of a line, and empty fields may be included for readability.

Capabilities in **printcap** are all introduced by two-character codes, and are of three types:

Boolean Capabilities that indicate that the printer has some particular feature. Boolean capabilities are simply written between the '**:**' characters, and are indicated by the word 'bool' in the **type** column of the capabilities table below.

Numeric Capabilities that supply information such as baud-rates, number of lines per page, and so on. Numeric capabilities are indicated by the word **num** in the **type** column of the capabilities table below. Numeric capabilities are given by the two-character capability code followed by the '#' character, followed by the numeric value. The following example is a numeric entry stating that this printer should run at 1200 baud:

:br#1200:

String Capabilities that give a sequence which can be used to perform particular printer operations such as cursor motion. String valued capabilities are indicated by the word **str** in the **type** column of the capabilities table below. String valued capabilities are given by the two-character capability code followed by an '=' sign and then a string ending at the next following '**:**'. For example,

:rp=spinwriter:

is a sample entry stating that the remote printer is named **spinwriter**.

Sun386i DESCRIPTION

On Sun386i systems, **lpr**(1) and related printing commands use the Network Information Service (NIS) to obtain the **printcap** entry for a named printer if the entry does not exist in the local **/etc/printcap** file. For example, when a user issues the command:

lpr -Pnewprinter foo

lpr searches **/etc/printcap** on the local system for an entry for **newprinter**. If no local entry for **newprinter** exists, then **lpr** searches the NIS map called **printcap**. The search is invisible to the user.

lpr creates the spooling directory for the printer automatically if no spooling directory exists.

System administrators can make a printer available to the entire NIS domain by placing an entry for that printer in the NIS **printcap** map, typically using **snap**. Otherwise, the system administrator must edit the **/etc/printcap** file on the NIS master and then rebuild the NIS map.

CAPABILITIES

Name	*Type*	*Default*	*Description*
af	str	NULL	name of accounting file
br	num	none	if lp is a tty, set the baud rate (ioctl call)
cf	str	NULL	cifplot data filter
df	str	NULL	TeX data filter (DVI format)
du	str	0	User ID of user 'daemon'.
fc	num	0	if lp is a tty, clear flag bits
ff	str	''\f''	string to send for a form feed
fo	bool	false	print a form feed when device is opened
fs	num	0	like 'fc' but set bits
gf	str	NULL	graph data filter (plot(3X) format)
hl	bool	false	print the burst header page last
ic	bool	false	driver supports (non standard) ioctl to indent printout
if	str	NULL	name of input/communication filter (created per job)
lf	str	''/dev/console''	error logging file name
lo	str	''lock''	name of lock file
lp	str	''/dev/lp''	device name to open for output
mc	num	0	maximum number of copies
ms	str	NULL	list of terminal modes to set or clear
mx	num	1000	maximum file size (in BUFSIZ blocks), zero = unlimited
nd	str	NULL	next directory for list of queues (unimplemented)
nf	str	NULL	ditroff data filter (device independent troff)
of	str	NULL	name of output/banner filter (created once)
pc	num	200	price per foot or page in hundredths of cents
pl	num	66	page length (in lines)
pw	num	132	page width (in characters)
px	num	0	page width in pixels (horizontal)
py	num	0	page length in pixels (vertical)
rf	str	NULL	filter for printing FORTRAN style text files
rg	str	NULL	restricted group. Only members of group allowed access
rm	str	NULL	machine name for remote printer
rp	str	''lp''	remote printer name argument
rs	bool	false	restrict remote users to those with local accounts
rw	bool	false	open printer device read/write instead of write-only
sb	bool	false	short banner (one line only)
sc	bool	false	suppress multiple copies
sd	str	''/var/spool/lpd''	spool directory
sf	bool	false	suppress form feeds
sh	bool	false	suppress printing of burst page header
st	str	''status''	status file name
tc	str	NULL	name of similar printer; must be last
tf	str	NULL	troff data filter (C/A/T phototypesetter)
tr	str	NULL	trailer string to print when queue empties
vf	str	NULL	raster image filter
xc	num	0	if lp is a tty, clear local mode bits
xs	num	0	like 'xc' but set bits

If the local line printer driver supports indentation, the daemon must understand how to invoke it.

Note: the **fs**, **fc**, **xs**, and **xc** fields are flag *masks* rather than flag *values*. Certain default device flags are set when the device is opened by the line printer daemon if the device is connected to a terminal port. The flags indicated in the **fc** field are then cleared; the flags in the **fs** field are then set (or vice-versa, depending on the order of **fc***#nnnn* and **fs***#nnnn* in the **/etc/printcap** file). The bits cleared by the **fc** field and set by the **fs** field are those in the **sg_flags** field of the **sgtty** structure, as set by the **TIOCSETP ioctl** call, and the bits cleared by the **xc** field and set by the **xs** field are those in the "local flags" word, as set by the **TIOCLSET ioctl** call. See **ttcompat**(4M) for a description of these flags. For example, to set exactly the flags 06300 in the **fs** field, which specifies that the **EVENP**, **ODDP**, and **XTABS** modes are to be set, and all other flags are to be cleared, do:

:fc#0177777:fs#06300:

The same process applies to the **xc** and **xs** fields. Alternatively, the **ms** field can be used to specify modes to be set and cleared. These modes are specified as **stty**(1V) modes; any mode supported by **stty** may be specified, except for the baud rate which must be specified with the **br** field. This permits modes not supported by the older terminal interface described in **ttcompat**(4M) to be set or cleared. Thus, to set the terminal port to which the printer is attached to even parity, TAB expansion, no NEWLINE to RETURN/LINEFEED translation, and RTS/CTS flow control enabled, do:

:ms=evenp,-tabs,nl,crtscts:

On Sun386i systems, the **tc** field, as in the **termcap**(5) file, must appear last in the list of capabilities. It is recommended that each type of printer have a general entry describing common capabilities; then an individual printer can be defined with its particular capabilities plus a **tc** field that points to the general entry for that type of printer.

FILES

/etc/printcap

SEE ALSO

lpq(1), **lpr**(1), **lprm**(1), **plot**(1G), **snap**(1), **stty**(1V), **plot**(3X), **ttcompat**(4M), **termcap**(5), **lpc**(8), **lpd**(8), **pac**(8)

System and Network Administration

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.

NAME

proto – prototype job file for at

SYNOPSIS

/var/spool/cron/.proto

/var/spool/cron/.proto.*queue*

DESCRIPTION

When a job is submitted to **at** or **batch**, (see **at**(1)) the job is constructed as a shell script. First, a prologue is constructed, consisting of:

- A header specifying the owner, job name, and shell that should be used to run the job, and a flag indicating whether mail should be sent when the job completes;
- A set of Bourne shell commands to make the environment (see **environ**(5V)) for the **at** job the same as the current environment;
- A command to run the user's shell (as specified by the **SHELL** environment variable) with the rest of the job file as input.

at then reads a "prototype file," and constructs the rest of the job file from it.

Text from the prototype file is copied to the job file, except for special "variables" that are replaced by other text:

$d	is replaced by the current working directory
$l	is replaced by the current file size limit (see **ulimit**(3C))
$m	is replaced by the current umask (see **umask**(2V))
$t	is replaced by the time at which the job should be run, expressed as seconds since January 1, 1970, 00:00 Greenwich Mean Time, preceded by a colon
$<	is replaced by text read by **at** from the standard input (that is, the commands provided to **at** to be run in the job)

If the job is submitted in queue *queue*, **at** uses the file **/var/spool/cron/.proto.***queue* as the prototype file if it exists, otherwise it will use the file **/var/spool/cron/.proto**.

EXAMPLES

The standard **.proto** file supplied with SunOS is:

```
#
# @(#)proto.5 1.3 89/10/05 SMI; from S5R3 1.1
#
cd $d
umask $m
$<
```

which causes commands to change the current directory in the job to the current directory at the time **at** was run, and to change the umask in the job to the umask at the time **at** was run, to be inserted before the commands in the job.

FILES

/var/spool/cron/.proto
/var/spool/cron/.proto.*queue*

SEE ALSO

at(1)

NAME

protocols – protocol name data base

SYNOPSIS

/etc/protocols

DESCRIPTION

The **protocols** file contains information regarding the known protocols used in the TCP/IP. For each protocol a single line should be present with the following information:

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.

EXAMPLE

The following example is taken from SunOS.

```
#
# Internet (IP) protocols
#
ip      0     IP      # internet protocol, pseudo protocol number
icmp    1     ICMP    # internet control message protocol
ggp     3     GGP     # gateway-gateway protocol
tcp     6     TCP     # transmission control protocol
pup     12    PUP     # PARC universal packet protocol
udp     17    UDP     # user datagram protocol
```

FILES

/etc/protocols

SEE ALSO

getprotoent(3N)

BUGS

A name server should be used instead of a static file. A binary indexed file format should be available for fast access.

NAME

publickey – public key database

SYNOPSIS

/etc/publickey

DESCRIPTION

/etc/publickey is the public key database used for secure networking. Each entry in the database consists of a network user name (which may either refer to 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 its login password (also in hex notation).

This file is altered either by the user through the **chkey**(1) command or by the system administrator through the **newkey**(8) command. The file **/etc/publickey** should only contain data on the Network Information Service (NIS) master machine, where it is converted into the NIS database **publickey.byname**.

The **/etc/publickey** file contains a default entry for **nobody**. If this entry is commented out, **chkey** only allows user to edit their existing entry, it will not allow them to create new entries.

SEE ALSO

chkey(1), **publickey**(3R), **newkey**(8), **ypupdated**(8C)

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.

NAME

queuedefs – queue description file for at, batch, and cron

SYNOPSIS

/var/spool/cron/queuedefs

DESCRIPTION

The **queuedefs** file describes the characteristics of the queues managed by **cron**(8). 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**(5) 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 more than 25 jobs were running in all the queues. The default value is 60.

Lines beginning with # are comments, and are ignored.

EXAMPLE

```
#
# @(#)queuedefs 1.1 87/02/18 SMI; from S5R3
#
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** jobs, can have up to 2 jobs running simultaneously; those jobs will be run with a **nice** value of 2. If a job cannot be run because too many other jobs are running, **cron** 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

/var/spool/cron/queuedefs

SEE ALSO

at(1), **nice**(1), **crontab**(5), **cron**(8)

NAME

rasterfile – Sun's file format for raster images

SYNOPSIS

#include <rasterfile.h>

DESCRIPTION

A rasterfile is composed of three parts: first, a header containing 8 integers; second, a (possibly empty) set of colormap values; and third, the pixel image, stored a line at a time, in increasing *y* order. The image is layed out in the file as in a memory pixrect. Each line of the image is rounded up to the nearest 16 bits.

The header is defined by the following structure:

```
struct rasterfile {
        int     ras_magic;
        int     ras_width;
        int     ras_height;
        int     ras_depth;
        int     ras_length;
        int     ras_type;
        int     ras_maptype;
        int     ras_maplength;
};
```

The *ras_magic* field always contains the following constant:

```
#define RAS_MAGIC     0x59a66a95
```

The *ras_width*, *ras_height*, and *ras_depth* fields contain the image's width and height in pixels, and its depth in bits per pixel, respectively. The depth is either 1 or 8, corresponding to standard frame buffer depths. The *ras_length* field contains the length in bytes of the image data. For an unencoded image, this number is computable from the *ras_width*, *ras_height*, and *ras_depth* fields, but for an encoded image it must be explicitly stored in order to be available without decoding the image itself. Note: the length of the header and of the (possibly empty) colormap values are not included in the value of the *ras_length* field; it is only the image data length. For historical reasons, files of type RT_OLD will usually have a 0 in the *ras_length* field, and software expecting to encounter such files should be prepared to compute the actual image data length if needed. The *ras_maptype* and *ras_maplength* fields contain the type and length in bytes of the colormap values, respectively. If *ras_maptype* is not RMT_NONE and the *ras_maplength* is not 0, then the colormap values are the *ras_maplength* bytes immediately after the header. These values are either uninterpreted bytes (usually with the *ras_maptype* set to RMT_RAW) or the equal length red, green and blue vectors, in that order (when the *ras_maptype* is RMT_EQUAL_RGB). In the latter case, the *ras_maplength* must be three times the size in bytes of any one of the vectors.

SEE ALSO

SunView Programmer's Guide

NAME

remote – remote host description file

SYNOPSIS

/etc/remote

DESCRIPTION

The systems known by **tip**(1C) and their attributes are stored in an ASCII file which is structured somewhat like the **termcap**(5) 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.

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 zero), **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**(5). A '**%**' sign in the telephone number indicates a 5-second delay for the Ventel Modem.

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.

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
/etc/phones

SEE ALSO

tip(1C), **phones**(5), **termcap**(5)

NAME

resolv.conf – configuration file for domain name system resolver

DESCRIPTION

The resolver configuration file contains information that is read by the domain name system resolver library the first time it is invoked in a process. It is only necessary to create this file to specify an explicit default domain name other than the default one derived from the **domainname**(1) command, or to specify name servers to use on other machines. 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

The different configuration options are:

nameserver *address* The Internet address (in dot notation) of a name server that the resolver should query. Up to MAXNS (currently 3) name servers may be listed. In that case the resolver library queries tries them in the order listed. The policy used is to try a name server, and if the query times out, try the next, until out of name servers, then repeat trying all the name servers until a maximum number of retries are made. If there are no **nameserver** lines in this file, then the loopback address is used, so there must be a name server running on the same machine.

domain *name* The default domain to append to names that do not have a dot in them, and used in searches. If there is no **domain** line in this file, then it is derived from the domain set by the **domainname**(1) command, usually by removing the first component. For example, if the **domainname**(1) is set to "foo.podunk.edu" then the default domain used by the resolver will be "podunk.edu". The is the same policy used by **sendmail**(8).

The keyword-value pair must appear on a single line, and the keyword (for instance, **nameserver**) must start the line. The value follows the keyword, separated by white space.

FILES

/etc/resolv.conf

SEE ALSO

domainname(1), **gethostent**(3N), **resolver**(3), **named**(8C), **nslookup**(8C), RFC 1034, RFC 1035

System and Network Administration

NAME

rfmaster – Remote File Sharing name server master file

SYNOPSIS

/usr/nserve/rfmaster

DESCRIPTION

The **rfmaster** file is an ASCII text file that identifies the hosts that are responsible for providing primary and secondary domain name service for Remote File Sharing domains. This file contains a series of entries, each terminated by a NEWLINE; a record may be extended over more than one line by escaping the NEWLINE with a backslash. Fields in each record are separated by white space. Each record has three fields: *name*, *type*, and *data*.

The *type* field, which defines the meaning of the *name* and *data* fields, has three possible values:

p Primary domain name server. In this case, *name* is the domain name and *data* is the full hostname of the primary name server, specified as:

domain.nodename

There can be only one primary name server per domain.

s Define a secondary name server for a domain. In this case, *name* and *data* are the same as for the **p** type. The order of the **s** entries in the **rfmaster** file determines the order in which secondary name servers take over when the current domain name server fails.

a Define a network address for a machine. In this case, *name* is the full domain name for the machine, and *data* is the network address. The network address can be in plain ASCII text or it can be preceded by a '\x' to be interpreted as hexadecimal notation.

There are at least two lines in the **rfmaster** file per domain name server: one **p** line and one **a** line. Together, they define the primary and its network address. There should also be at least one secondary name server in each domain.

This file is created and maintained on the primary domain name server. When a machine other than the primary tries to start Remote File Sharing, this file is read to determine the address of the primary. If this file is missing, the **–p** option of **rfstart** must be used to identify the primary. After that, a copy of the primary's **rfmaster** file is automatically placed on the machine.

Domains not served by the primary can also be listed in the **rfmaster** file. By adding primary, secondary, and address information for other domains on a network, machines served by the primary will be able to share resources with machines in other domains.

A primary name server may be a primary for more than one domain. However, the secondaries must then also be the same for each domain served by the primary.

EXAMPLE

An example of an **rfmaster** file is shown below. The network addresses given in the example are IP addresses; for more information on their format and how to generate them, see **hostrfs**(8).

```
sunrfs          p    sunrfs.estale
sunrfs          s    sunrfs.ivy
sunrfs.estale   a    \x000214508190320d
sunrfs.ivy      a    \x0002145081903246
```

Note: If a line in the **rfmaster** file begins with a '#' (pound sign) character, the entire line will be treated as a comment.

FILES

/usr/nserve/rfmaster

SEE ALSO

rfstart(8)

System and Network Administration

NAME

rgb – available colors (by name) for coloredit

SYNOPSIS

.rgb

$HOME/.rgb

/usr/lib/.rgb

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

.rgb is an ASCII file containing consecutive lines terminated by newlines. Each line starts with three integers, each in the range 0-255. These integers are the RGB equivalent for the color named on the same line. At least one tab character delimits the last integer from the name field. The coloreditor searches for this file, first in the current directory; next, in the users home directory; and finally, in **/usr/lib**. The user can add to or delete from the **.rgb** file that he or she has access to, thus changing the available color table for subsequent invocations of coloredit.

EXAMPLES

The following is an example of a **.rgb** file.

0 0 0	**Black**
0 0 255	**Blue**
95 159 159	**Cadet Blue**
66 66 111	**Cornflower Blue**
107 35 142	**Dark Slate Blue**

SEE ALSO

coloredit(1)

NAME

rootmenu – root menu specification for SunView

SYNOPSIS

~/.rootmenu
/usr/lib/.rootmenu

DESCRIPTION

If a **.rootmenu** file is present in a user's home directory, it specifies the SunView menu, the menu that appears when the user clicks and holds the right mouse button in the background of the SunView desktop. If a **.rootmenu** file is not present in the user's home directory, **/usr/lib/.rootmenu** specifies the SunView menu.

Each line of a **.rootmenu** file has the format:

menu item *command*

menu item can be a character string, or an icon file delimited by angle brackets:

<icon-filename>

If *menu item* is a character string with embedded spaces, it must be enclosed by double quotes ('"').

command can be a command line to be executed when the menu item is selected, or one of the following reserved-word commands:

EXIT	Exit **sunview** (requires confirmation).
REFRESH	Redraw the entire screen.
MENU	This menu item is a pull-right item with a submenu. If a full pathname follows the **MENU** command, the submenu contents are taken from that file. Otherwise, all the lines between a **MENU** command and a matching **END** command are added to the submenu.
END	Mark the end of a nested submenu. The left side of this line should match the left side of a line with a **MENU** command.

If *command* is not one of the reserved-word commands, it is treated as a command line, although no shell interpretation is done.

Lines beginning with a '#' character are considered comments and are ignored.

If a user's **.rootmenu** file is modified, the SunView menu immediately reflects the changes.

See **sunview**(1) for more details about **.rootmenu**.

EXAMPLES

The following is a sample **.rootmenu** file:

```
#
#       sample root menu
#
"Lock Screen"           lockscreen
Tools   MENU
        Perfmeter               perfmeter
        Calculator              calc
        Mailtool                mailtool
Tools   END
"ShellTool"             shelltool
"CommandTool"           cmdtool
"Console"               cmdtool -C
#"MailTool"             mailtool
"TextEditor"            textedit
```

```
        "DefaultsEditor"        defaultsedit
        #"IconEditor"           iconedit
        #"DbxTool"              dbxtool
        "PerfMeter"             perfmeter
        #"GraphicsTool"         gfxtool
        "Redisplay All"         REFRESH
        "Exit Suntools"         EXIT
```

SEE ALSO

sunview(1)

NAME

rpc – rpc program number data base

SYNOPSIS

/etc/rpc

DESCRIPTION

The **rpc** file contains user readable names that can be used in place of rpc program numbers. Each line has the following information:

rpc-program-server *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.

EXAMPLE

Here is an example of the /etc/rpc file from the SunOS System.

```
#
#       rpc  1.10  87/04/10
#
portmapper      100000    portmap sunrpc
rstatd          100001    rstat rup perfmeter
rusersd         100002    rusers
nfs             100003    nfsprog
ypserv          100004    ypprog
mountd          100005    mount showmount
ypbind          100007
walld           100008    rwall shutdown
yppasswdd       100009    yppasswd
etherstatd      100010    etherstat
rquotad         100011    rquotaprog quota rquota
sprayd          100012    spray
3270_mapper     100013
rje_mapper      100014
selection_svc   100015    selnsvc
database_svc    100016
rexd            100017    rex
alis            100018
sched           100019
llockmgr        100020
nlockmgr        100021
x25.inr         100022
statmon         100023
status          100024
bootparam       100026
ypupdated       100028    ypupdate
keyserv         100029    keyserver
```

FILES

/etc/rpc

SEE ALSO

getrpcent(3N)

NAME

sccsfile – format of an SCCS history file

DESCRIPTION

An SCCS file is an ASCII file consisting of six logical parts:

checksum	character count used for error detection
delta table	log containing version info and statistics about each delta
usernames	login names and/or group IDs of users who may add deltas
flags	definitions of internal keywords
comments	arbitrary descriptive information about the file
body	the actual text lines intermixed with control lines

Each section is described in detail below.

Conventions

Throughout an SCCS file there are lines which begin with the ASCII SOH (start of heading) character (octal 001). This character is hereafter referred to as the *control character*, and will be represented as '^A'. If a line described below is not depicted as beginning with the control character, it cannot do so and still be within SCCS file format.

Entries of the form *ddddd* represent a five digit string (a number between 00000 and 99999).

Checksum

The checksum is the first line of an SCCS file. The form of the line is:

^A **h***ddddd*

The value of the checksum is the sum of all characters, except those contained in the first line. The ^A**h** provides a *magic number* of (octal) 064001.

Delta Table

The delta table consists of a variable number of entries of the form:

^A**s** *inserted/deleted/unchanged*
^A**d** *type sid yr/mo/da hr:mi:se username serial-number predecessor-sn*
^A**i** *include-list*
^A**x** *exclude-list*
^A**g** *ignored-list*
^A**m** *mr-number*
...
^A**c** *comments ...*
...
^A**e**

The first line (^A**s**) contains the number of lines inserted/deleted/unchanged respectively. The second line (^A**d**) contains the type of the delta (normal: **D**, and removed: **R**), the SCCS ID of the delta, the date and time of creation of the delta, the user-name corresponding to the real user ID at the time the delta was created, and the serial numbers of the delta and its predecessor, respectively.

The ^A**i**, ^A**x**, and ^A**g** lines contain the serial numbers of deltas included, excluded, and ignored, respectively. These lines do not always appear.

The ^A**m** lines (optional) each contain one MR number associated with the delta; the ^A**c** lines contain comments associated with the delta.

The ^A**e** line ends the delta table entry.

User Names

The list of user-names and/or numerical group IDs of users who may add deltas to the file, separated by NEWLINE characters. The lines containing these login names and/or numerical group IDs are surrounded by the bracketing lines ^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 %T% 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 %M% ID keyword.

^Af f *floor*
Defines the "floor" release; the release below which no deltas may be added.

^Af c *ceiling*
Defines the "ceiling" release; the release above which no deltas may be added.

^Af d *default-sid*
The **d** flag defines the default SID to be used when none is specified on an SCCS **get** command.

^Af 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).

^Af j Enables the SCCS **get** command to allow concurrent edits of the same base SID.

^Af l *lock-releases*
Defines a *list* of releases that are locked against editing.

^Af q user defined
Defines the replacement for the %Q% ID keyword.

^Af 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 ^At and ^AT. 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(1), **sccs-admin**(1), **sccs-cdc**(1), **sccs-comb**(1), **sccs-delta**(1), **sccs-get**(1), **sccs-help**(1), **sccs-prs**(1), **sccs-prt**(1), **sccs-rmdel**(1), **sccs-sact**(1), **sccs-sccsdiff**(1), **sccs-unget**(1), **sccs-val**(1), **what**(1)

NAME

services – Internet services and aliases

DESCRIPTION

The **services** file contains an entry for each service available through the TCP/IP. Each entry consists of a line of 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 spaces or TAB's. A '#' (pound-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/services

SEE ALSO

getservent(3N), **inetd.conf**(5)

BUGS

A name server should be used instead of a static file.

NAME

setup.pc – master configuration file for DOS

SYNOPSIS

~/pc/setup.pc

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

The **setup.pc** file in your home PC directory, **~/pc**, is the master configuration file for DOS. Changes to the file take effect for all new DOS windows you start. The definitions made in **setup.pc** and **AUTOEXEC.BAT** serve to define your system to DOS. Among other things, the **setup.pc** file defines:

- The printers or devices to which you assign DOS printer names (LPT1, LPT2, LPT3)
- The devices or boards that are tied to the DOS communications devices (COM1, COM2)
- The name of a special DOS quick-start file that you may have set up
- The drive C file to be used

The format of each line is as follows; separators can be TAB or SPACE characters:

DOS Device *SunOS Device or Command*

DOS Device

The name of the device as DOS knows it. For example, the device name for the first diskette drive in DOS is "A".

SunOS Device or Command

The name of the device as the SunOS system knows it. This can also be a symbolic link to the real device name. For example, **/etc/dos/defaults/diskette_a** is a symbolic link to **/dev/rfd0c**. For emulated DOS printers (LPT1, LPT2, or LPT3), specify a command or command pipeline.

EXAMPLES

```
# DOS Device    SunOS Device Path Name
#
A               /etc/dos/defaults/diskette_a
#B              /etc/dos/defaults/diskette_b
C               ~/pc/C:
COM1            /etc/dos/defaults/com1
#COM2           /etc/dos/defaults/com2
LPT1            lpr
LPT2            cat >>~/lpt-2
LPT3            psfx80 | lpr
SAVE            ~/pc/.quickpc
#CMDTOOL
#TEXT
#BOARDS
```

Placed at the beginning of a line to indicate a comment.

A, B Diskette drivers defined using the standard SunOS names for the Sun386i diskette drives. Drive **A** is normally assigned to the built-in diskette drive.

C The emulated C drive. It is actually stored as one large system file.

COM1, COM2
Serial ports. The first DOS serial port (COM1) is assigned to the Sun386i built-in serial port. To use the built-in serial port as COM2, comment out the COM1 line and uncomment the COM2 line. (DOS Windows directs the output of either COM1 or COM2 to the built-in port, but uses different interrupt levels so that COM2 "appears" to DOS to be a second serial port.) You can also add a real second serial port by installing an AT or XT card and enabling the SunOS ATS driver.

LPT1, LPT2, LPT3
Emulated printers. DOS printer names can be assigned to SunOS printers, other devices, or files.

SAVE The "quick-start" file DOS reads at startup for faster loading.

CMDTOOL
Used to list the SunOS commands that must run in a separate Commands window when started from DOS. The following SunOS commands automatically run in a Commands window when you run them from DOS:

mail man more passwd rlogin stty vi

If there are other SunOS commands or applications you want to run from DOS, and these commands require keyboard entry or Commands window display, list them here. If you add entries to this line, separate them with a SPACE character, and be sure to remove the # (comment) symbol to activate the line.

TEXT Specifies a list of "text-only" DOS programs. Such programs do not require a PC window because they do not print at specific screen positions; they can print text in a current Commands window if that is where you are working at the time. An example is a C compiler or a linker that runs from the DOS command line. If you place entries on this line, separate them with a SPACE character, and be sure to remove the # symbol to activate the text-only line.

BOARDS
A list of boards that DOS should attempt to activate when opening a DOS window. Each board you list here must have a corresponding entry in the **boards.pc file** (see **boards.pc**(5)).

You can create task-specific DOS environments by setting up additional **setup.pc** files to attach different printers, drive C files, and other real and emulated devices.

If you are installing a board that duplicates a function normally enabled in the **setup.pc** file, you should disable the corresponding **setup.pc** line by commenting it out with **#**.

FILES

~/pc/setup.pc	Personal **setup.pc** file, copied to the user's **pc** directory when DOS is started for the first time.
/etc/dos/defaults/setup.pc	Master copy of **setup.pc** for the workstation.

SEE ALSO

dos(1), **boards.pc**(5)

Sun386i User's Guide,
Sun386i Advanced Skills,
Sun MS-DOS Reference Manual

NAME

sm, sm.bak, sm.state – in.statd directory and file structures

SYNOPSIS

/etc/sm, /etc/sm.bak, /etc/sm.state

DESCRIPTION

/etc/sm and **/etc/sm.bak** are directories generated by **in.statd**. Each entry in **/etc/sm** represents the name of the machine to be monitored by the **in.statd** daemon. Each entry in **/etc/sm.bak** represents the name of the machine to be notified by the **in.statd** daemon upon its recovery.

/etc/sm.state is a file generated by **rpc.statd** to record the its version number. This version number is incremented each time a crash or recovery takes place.

FILES

/etc/sm
/etc/sm.bak
/etc/sm.state

SEE ALSO

lockd(8C), **statd**(8C)

NAME

sm, sm.bak, state – statd directories and file structures

SYNOPSIS

/etc/sm /etc/sm.bak /etc/state

DESCRIPTION

/etc/sm and **/etc/sm.bak** are directories generated by **statd**. Each entry in **/etc/sm** represents the name of the machine to be monitored by the **statd** daemon. Each entry in **/etc/sm.bak** represents the name of the machine to be notified by the **statd** daemon upon its recovery.

/etc/state is a file generated by **statd** to record its version number. This version number is incremented each time a crash or recovery takes place.

FILES

/etc/sm
/etc/sm.bak
/etc/state

SEE ALSO

lockd(8C), **statd**(8C)

NAME

sunview – initialization file for SunView

SYNOPSIS

~/.sunview
~/.suntools
/usr/lib/.sunview

DESCRIPTION

If the file **.sunview** or **.suntools** is present in a user's home directory when the user starts up **sunview**(1), **sunview** starts up the "tools", or window-based applications listed in this file. You can use a **.sunview** or **.suntools** file to customize your desktop layout. If a **.sunview** or **.suntools** file is not present in the user's home directory, **sunview** starts up the tools listed in **/usr/lib/.sunview**.

Each line of a **.sunview** or **.suntools** file has the format:

SunView-tool [*options*]

SunView-tool is in the form of a command line, although no shell interpretation is done. *options* are command line options which may include SunView generic tool arguments (see **sunview**(1) for a description of generic tool arguments). Lines beginning with the '#' character are considered comments and are ignored.

EXAMPLES

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

```
#
#       sample .sunview file
#
cmdtool   -Wp  0  0 -WP   0  0 -Wh  3 -Ww 80 -Wl "<< CONSOLE >>" -WL "console" -C
clock     -Wp 497  32 -WP  704   0 -Wi -Wh 1
cmdtool   -Wp  0  71 -WP  772   0 -Wi -Wh 44 -Ww 80
textedit  -Wp 259  98 -WP  840   0 -Wi
mailtool  -Wp 492  71 -WP  908   0 -Wi
```

SEE ALSO

sunview(1), **toolplaces**(1)

NAME

svdtab – SunView device table

SYNOPSIS

/etc/svdtab

DESCRIPTION

The **/etc/svdtab** contains information that is used by the window system (for example, **sunview**(1)) to change the owner, group, and permissions of the window devices (**/dev/win***) upon startup. By default *all* lines in this file are commented out. That is, all security is disabled. To enable security, uncomment the following line in **/etc/svdtab** and start up the window system again:

#0600

If **/etc/svdtab** contains an entry, the owner and group of the **win** devices are set to the owner and group of the console. The permissions are set as specified in **/etc/svdtab**. The recommended permissions for normal security is **0600**.

Once the window devices are owned by the user, their permissions and ownership can be changed using **chmod**(1V) and **chown**(8), as with any user-other file.

EXAMPLES

The following is an example entry of the **/etc/svdtab** file:

0600

This entry specifies that upon SunView startup, the owner, group and permissions of **/dev/win*** will be set to the user's username, the user's group and **0600**, respectively. Upon exiting the window system, the owner and group of **/dev/win***, will be reset to **root**, and **wheel**. The permissions remain as set in **/etc/svdtab**. If no entry appears in this file, the owner, group and permissions will *not* be changed.

SEE ALSO

chmod(1V), **sunview**(1), **chown**(8)

NOTES

If the window system dies unnaturally, for example by **kill**(1), the owner, group and permissions remain as set when the window was started up.

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**(8), to forward a system message to appropriate log files and/or users. **syslog** preprocesses this file through **m4**(1V) to obtain the correct information for certain log files.

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 **ftpd**(8C), **routed**(8C), etc.

auth The authorization system: **login**(1), **su**(1V), **getty**(8), etc.

lpr The line printer spooling system: **lpr**(1), **lpc**(8), **lpd**(8), etc.

news Reserved for the USENET network news system.

uucp Reserved for the UUCP system; it does not currently use the **syslog** mechanism.

cron The **cron/at** facility; **crontab**(1), **at**(1), **cron**(8), etc.

local0-7 Reserved for local use.

mark For timestamp messages produced internally by **syslogd**.

***** An asterisk indicates all facilities except for the **mark** facility.

Recognized values for *level* are (in descending order of severity):

emerg For panic conditions that would normally be broadcast to all users.

alert For conditions that should be corrected immediately, such as a corrupted system database.

crit For warnings about critical conditions, such as hard device errors.

err For other errors.

warning For warning messages.

notice For conditions that are not error conditions, but may require special handling.

info Informational messages.

debug For messages that are normally used only when debugging a program.

none Do not send messages from the indicated *facility* to the selected file. For example, a *selector* of

***.debug;mail.none**

will send all messages *except* mail messages to the selected file.

The *action* field indicates where to forward the message. Values for this field can have one of four forms:

- A filename, beginning with a leading slash, which indicates that messages specified by the *selector* are to be written to the specified file. The file will be opened in append mode.
- The name of a remote host, prefixed with an **@**, as with: **@***server*, which indicates that messages specified by the *selector* are to be forwarded to the **syslogd** on the named host.
- 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 character is a '#' are treated as comments.

Sun386i DESCRIPTION

The file is as described above, except that there is an additional valid entry type, for translation. A line containing the keyword "translate," if present, specifies how system error messages are translated, suppressed, or forwarded to appropriate log files and/or users.

A translation entry in the file is composed of five TAB-separated fields:

translate *source* *facility* *input* *output*

The *translate* field consists of the word **translate** and is used to indicate that this is a translation entry.

The *source* field contains a comma separated list of source names. Recognized sources are:

klog Messages placed in **/dev/klog** by the kernel.

log Messages placed in **/dev/log** file by local programs.

syslog Messages placed in the internet socket by programs on other systems.

***** An asterisk indicates all three sources (**klog**, **log** and **syslog**).

The *facility* field contains a comma-separated list of facilities.

The *input* field is the name of the file used to map error messages (in printf format strings) to numbers. This number is used to locate a new string in the file specified in the output field. The format of both files is described in **translate**(5).

The output file specified by the output field translates the numbers from the input file into the desired error messages, and also specifies the format to be used to output each message.

EXAMPLE

With the following configuration file:

```
*.notice;mail.info      /var/log/notice
*.crit                  /var/log/critical
kern,mark.debug         /dev/console
kern.err                @server
*.emerg                 *
*.alert                 root,operator
*.alert;auth.warning    /var/log/auth
```

syslogd will log all mail system messages except **debug** messages and all **notice** (or higher) messages into a file named **/var/log/notice**. It logs all critical messages into **/var/log/critical**, and all kernel messages and 20-minute marks onto the system console.

Kernel messages of **err** (error) severity or higher are forwarded to the machine named *server*. Emergency messages are forwarded to all users. The users "root" and "operator" are informed of any **alert** messages. All messages from the authorization system of **warning** level or higher are logged in the file **/var/log/auth**.

FILES

/etc/syslog.conf
/var/log/notice
/var/log/critical
/var/log/auth

SEE ALSO

at(1), **crontab**(1), **logger**(1), **login**(1), **lpr**(1), **m4**(1V), **su**(1V), **syslog**(3), **translate**(5), **cron**(8), **ftpd**(8C), **getty**(8), **lpc**(8), **lpd**(8), **routed**(8C), **syslogd**(8)

NAME

systems – NIS systems file

SYNOPSIS

/etc/systems

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

The **/etc/systems** file is used only by SNAP and Automatic System Installation, and contains basic information about each host on the network. It is an ASCII file in the /etc directory on the Network Information Service (NIS) master server. To successfully administer all systems in a NIS domain using SNAP, there must be an entry in this file for each host listed in the **/etc/hosts** file. For each host, this file contains a one-line entry, of the following form, where each field *must* be separated by a TAB character:

system architecture sunos "hostid" memory_size disk_size network_role

system is the name of a host, whether it is on a network or a standalone system. This field contains only lowercase and numeric characters, must start with a lower-case character, and must not be longer than 32 characters.

architecture
indicates the architecture of the specified system. This can be **s386**, **sun4**, **sun3**, **sun2**, **sun1**, **pcnfs**, or **other**.

sunos indicates the SunOS operating system version the system is running. Typically, the form is **sunos***version_number* or **unknown**. SNAP always inserts **unknown** when adding new systems.

hostid the system host ID, as obtained from **/bin/hostid**. This entry must be in quotes. If the host ID is **unknown**, an empty string (" ") is specified. SNAP always inserts an empty string when adding new systems.

memory_size
amount of memory, in kilobytes. This can be **8000** (for 8 megabytes), **4000** (for 4 megabytes), or **–1** for **unknown**. SNAP always inserts **–1** when adding new systems.

disk_size
amount of disk space, in kilobytes. This can be any value, but typically should be close to the actual disk size or to the total amount of disk space, if expansion disks were added. Diskless clients would have a zero value, while **unknown** disk sizes are specified by a **–1** value. SNAP always inserts **–1** when adding new network clients.

network_role
indicates the role the system plays on the network. This can be **master_bootserver**, **slave_bootserver**, **network_client**, or **diskless_client**.

EXAMPLES

Here is a sample systems file:

```
vulcan      s386    sunos4.0.1    "12345678"    8000    327000    master_bootserver
polaris     s386    sunos4.0.1    ""            8000    91000     slave_bootserver
star        sun4    sunos4.1      ""            8000    91000     network_client
traveler    s386    sunos4.0.1    ""            8000    0         diskless_client
```

FILES

/etc/systems
/etc/hosts
/bin/hostid

SEE ALSO

snap(1), **vipw**(8)

System and Network Administration,
Sun386i Advanced Administration

NOTES

Take precautions to lock the **/etc/systems** file against simultaneous changes if it will be edited with a text editor; editing with **vipw**(8) provides the necessary locking.

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

tar – tape archive file format

DESCRIPTION

tar, (the tape archive command) dumps several files into one, in a medium suitable for transportation.

A ''tar tape'' or file is a series of blocks. Each block is of size TBLOCK. A file on the tape is represented by a header block which describes the file, followed by zero or more blocks which give the contents of the file. At the end of the tape are two blocks filled with binary zeros, as an EOF indicator.

The blocks are grouped for physical I/O operations. Each group of *n* blocks (where *n* is set by the **b** keyletter on the **tar**(1) command line — default is 20 blocks) is written with a single system call; on nine-track tapes, the result of this write is a single tape record. The last group is always written at the full size, so blocks after the two zero blocks contain random data. On reading, the specified or default group size is used for the first read, but if that read returns less than a full tape block, the reduced block size is used for further reads, unless the **B** keyletter is used.

The header block looks like:

```
#define TBLOCK512
#define NAMSIZ 100
union hblock {
        char dummy[TBLOCK];
        struct header {
                char name[NAMSIZ];
                char mode[8];
                char uid[8];
                char gid[8];
                char size[12];
                char mtime[12];
                char chksum[8];
                char linkflag;
                char linkname[NAMSIZ];
        } dbuf;
};
```

name is a null-terminated string. The other fields are zero-filled octal numbers in ASCII. Each field (of width *w*) contains w-2 digits, a SPACE, and a null character, except *size* and *mtime*, which do not contain the trailing null. *name* is the name of the file, as specified on the **tar** command line. Files dumped because they were in a directory which was named in the command line have the directory name as prefix and */filename* as suffix. *mode* is the file mode, with the top bit masked off. *uid* and *gid* are the user and group numbers which own the file. *size* is the size of the file in bytes. Links and symbolic links are dumped with this field specified as zero. *mtime* is the modification time of the file at the time it was dumped. *chksum* is a decimal ASCII value which represents the sum of all the bytes in the header block. When calculating the checksum, the *chksum* field is treated as if it were all blanks. *linkflag* is ASCII '0' if the file is ''normal'' or a special file, ASCII '1' if it is an hard link, and ASCII '2' if it is a symbolic link. The name linked-to, if any, is in *linkname*, with a trailing null character. Unused fields of the header are binary zeros (and are included in the checksum).

The first time a given inode number is dumped, it is dumped as a regular file. The second and subsequent times, it is dumped as a link instead. Upon retrieval, if a link entry is retrieved, but not the file it was linked to, an error message is printed and the tape must be manually re-scanned to retrieve the linked-to file.

The encoding of the header is designed to be portable across machines.

SEE ALSO

tar(1)

BUGS

Names or linknames longer than NAMSIZ produce error reports and cannot be dumped.

NAME

term – terminal driving tables for nroff

SYNOPSIS

/usr/lib/term/tab*name*

DESCRIPTION

nroff(1) uses driving tables to customize its output for various types of output devices, such as terminals, line printers, daisy-wheel printers, or special output filter programs. These driving tables are written as C programs, compiled, and installed in the directory **/usr/lib/term**. The *name* of the output device is specified with the **–T** option of **nroff**. The structure of the terminal table is as follows:

```
#define    INCH    240

struct {
        int bset;
        int breset;
        int Hor;
        int Vert;
        int Newline;
        int Char;
        int Em;
        int Halfline;
        int Adj;
        char *twinit;
        char *twrest;
        char *twnl;
        char *hlr;
        char *hlf;
        char *flr;
        char *bdon;
        char *bdoff;
        char *ploton;
        char *plotoff;
        char *up;
        char *down;
        char *right;
        char *left;
        char *codetab[256–32];
        char *zzz;
} t;
```

The meanings of the various fields are as follows:

bset Bits to set in the **sg_flags** field of the **sgtty** structure before output; see **ttcompat**(4M).

breset Bits to reset in the **sg_flags** field of the **sgtty** structure after output; see **ttcompat**(4M).

Hor Horizontal resolution in fractions of an inch.

Vert Vertical resolution in fractions of an inch.

Newline Space moved by a NEWLINE (LINEFEED) character in fractions of an inch.

Char Quantum of character sizes, in fractions of an inch (that is, a character is a multiple of **Char** units wide).

Em Size of an em in fractions of an inch.

Halfline Space moved by a half-LINEFEED (or half-reverse-LINEFEED) character in fractions of an inch.

Adj Quantum of white space, in fractions of an inch. (that is, white spaces are a multiple of **Adj** units wide)

Note: if this is less than the size of the SPACE character (in units of **Char**; see below for how the sizes of characters are defined), **nroff** will output fractional SPACE characters using plot mode. Also, if the **–e** switch to **nroff** is used, **Adj** is set equal to **Hor** by **nroff**.

twinit Set of characters used to initialize the terminal in a mode suitable for **nroff**.

twrest Set of characters used to restore the terminal to normal mode.

twnl Set of characters used to move down one line.

hlr Set of characters used to move up one-half line.

hlf Set of characters used to move down one-half line.

flr Set of characters used to move up one line.

bdon Set of characters used to turn on hardware boldface mode, if any.

bdoff Set of characters used to turn off hardware boldface mode, if any.

ploton Set of characters used to turn on hardware plot mode (for Diablo type mechanisms), if any.

plotoff Set of characters used to turn off hardware plot mode (for Diablo type mechanisms), if any.

up Set of characters used to move up one resolution unit (**Vert**) in plot mode, if any.

down Set of characters used to move down one resolution unit (**Vert**) in plot mode, if any.

right Set of characters used to move right one resolution unit (**Hor**) in plot mode, if any.

left Set of characters used to move left one resolution unit (**Hor**) in plot mode, if any.

codetab Definition of characters needed to print an **nroff** character on the terminal. The first byte is the number of character units (**Char**) needed to hold the character; that is, **\001** is one unit wide, **\002** is two units wide, etc. The high-order bit (0200) is on if the character is to be underlined in underline mode (**.ul**). The rest of the bytes are the characters used to produce the character in question. If the character has the sign (0200) bit on, it is a code to move the terminal in plot mode. It is encoded as:

0100 bit on	vertical motion.
0100 bit off	horizontal motion.
040 bit on	negative (up or left) motion.
040 bit off	positive (down or right) motion.
037 bits	number of such motions to make.

zzz A zero terminator at the end.

All quantities which are in units of fractions of an inch should be expressed as '**INCH***num*/*denom*', where *num* and *denom* are respectively the numerator and denominator of the fraction; that is, 1/48 of an inch would be written as '**INCH/48**'.

If any sequence of characters does not pertain to the output device, that sequence should be given as a null string.

The following is a sample **codetab** encoding.

```
"\001 ",                    /*space*/
"\001!",                    /*!*/
"\001\"",                   /*"*/
"\001#",                    /*#*/
"\001$",                    /*$*/
```

```
"\001%",        /*%*/
"\001&",        /*&*/
"\001'",        /*'*/
"\001(",        /*(*/
"\001)",        /*)*/
"\001*",        /***/
"\001+",        /*+*/
"\001,",        /*,*/
"\001-",        /*-*/
"\001.",        /*.*/
"\001/",        /*/*/
"\2010",        /*0*/
"\2011",        /*1*/
"\2012",        /*2*/
"\2013",        /*3*/
"\2014",        /*4*/
"\2015",        /*5*/
"\2016",        /*6*/
"\2017",        /*7*/
"\2018",        /*8*/
"\2019",        /*9*/
"\001:",        /*:*/
"\001;",        /*;*/
"\001<",        /*<*/
"\001=",        /*=*/
"\001>",        /*>*/
"\001?",        /*?*/
"\001@",        /*@*/
"\201A",        /*A*/
"\201B",        /*B*/
"\201C",        /*C*/
"\201D",        /*D*/
"\201E",        /*E*/
"\201F",        /*F*/
"\201G",        /*G*/
"\201H",        /*H*/
"\201I",        /*I*/
"\201J",        /*J*/
"\201K",        /*K*/
"\201L",        /*L*/
"\201M",        /*M*/
"\201N",        /*N*/
"\201O",        /*O*/
"\201P",        /*P*/
"\201Q",        /*Q*/
"\201R",        /*R*/
"\201S",        /*S*/
"\201T",        /*T*/
"\201U",        /*U*/
"\201V",        /*V*/
"\201W",        /*W*/
"\201X",        /*X*/
"\201Y",        /*Y*/
```

```
"\201Z",                /*Z*/
"\001[",                /*[*/
"\001\\",               /*\*/
"\001]",                /*]*/
"\001^",                /*^*/
"\001_",                /*_*/
"\001`",                /*`*/
"\201a",                /*a*/
"\201b",                /*b*/
"\201c",                /*c*/
"\201d",                /*d*/
"\201e",                /*e*/
"\201f",                /*f*/
"\201g",                /*g*/
"\201h",                /*h*/
"\201i",                /*i*/
"\201j",                /*j*/
"\201k",                /*k*/
"\201l",                /*l*/
"\201m",                /*m*/
"\201n",                /*n*/
"\201o",                /*o*/
"\201p",                /*p*/
"\201q",                /*q*/
"\201r",                /*r*/
"\201s",                /*s*/
"\201t",                /*t*/
"\201u",                /*u*/
"\201v",                /*v*/
"\201w",                /*w*/
"\201x",                /*x*/
"\201y",                /*y*/
"\201z",                /*z*/
"\001{",                /*{*/
"\001|",                /*|*/
"\001}",                /*}*/
"\001~",                /*~*/
"\000\0",               /*narrow sp*/
"\001-",                /*hyphen*/
"\001\016Z\017",        /*bullet*/
"\002[]",               /*square*/
"\002--",               /*3/4 em dash*/
"\001_",                /*rule*/
"\0031/4",              /*1/4*/
"\0031/2",              /*1/2*/
"\0033/4",              /*3/4*/
"\001-",                /*minus*/
"\202fi",               /*fi*/
"\202fl",               /*fl*/
"\202ff",               /*ff*/
"\203ffi",              /*ffi*/
"\203ffl",              /*ffl*/
"\001\016p\017",        /*degree*/
```

```
"\001|\b\342-\302",                    /*dagger*/
"\001\301s\343s\302",                  /*section*/
"\001'",                               /*foot mark*/
"\001\033Z",                           /*acute accent*/
"\001`",                               /*grave accent*/
"\001_",                               /*underrule*/
"\001/",                               /*long slash*/
"\000\0",                              /*half narrow space*/
"\001 ",                               /*unpaddable space*/
"\001\016A\017",                       /*alpha*/
"\001\016B\017",                       /*beta*/
"\001\016C\017",                       /*gamma*/
"\001\016D\017",                       /*delta*/
"\001\016E\017",                       /*epsilon*/
"\001\016F\017",                       /*zeta*/
"\001\016G\017",                       /*eta*/
"\001\016H\017",                       /*theta*/
"\001\016I\017",                       /*iota*/
"\001\016J\017",                       /*kappa*/
"\001\016K\017",                       /*lambda*/
"\001\016L\017",                       /*mu*/
"\001\016M\017",                       /*nu*/
"\001\016N\017",                       /*xi*/
"\001\016O\017",                       /*omicron*/
"\001\016P\017",                       /*pi*/
"\001\016Q\017",                       /*rho*/
"\001\016R\017",                       /*sigma*/
"\001\016S\017",                       /*tau*/
"\001\016T\017",                       /*upsilon*/
"\001\016U\017",                       /*phi*/
"\001\016V\017",                       /*chi*/
"\001\016W\017",                       /*psi*/
"\001\016X\017",                       /*omega*/
"\001\016#\017",                       /*Gamma*/
"\001\016$\017",                       /*Delta*/
"\001\016(\017",                       /*Theta*/
"\001\016+\017",                       /*Lambda*/
"\001\016.\017",                       /*Xi*/
"\001\0160\017",                       /*Pi*/
"\001\0169\017",                       /*Sigma*/
"\000",                                /**/
"\001\0164\017",                       /*Upsilon*/
"\001\0165\017",                       /*Phi*/
"\001\0167\017",                       /*Psi*/
"\001\0168\017",                       /*Omega*/
"\001\016[\017",                       /*square root*/
"\001\016Y\017",                       /*\(ts yields script-l*/
"\001\016k\017",                       /*root en*/
"\001>\b_",                            /*>=*/
"\001<\b_",                            /*<=*/
"\001=\b_",                            /*identically equal*/
"\001-",                               /*equation minus*/
"\001\016o\017",                       /*approx =*/
```

```
"\001\016n\017",                                    /*approximates*/
"\001=\b/",                                         /*not equal*/
"\002-\242-\202>",                                  /*right arrow*/
"\002<\b\202-\242\200-",                            /*left arrow*/
"\001|\b^",                                         /*up arrow*/
"\001|\b\302v\342",                                 /*down arrow*/
"\001=",                                            /*equation equal*/
"\001\016|\017",                                    /*multiply*/
"\001\016}\017",                                    /*divide*/
"\001\016j\017",                                    /*plus-minus*/
"\001\243|\203_\203|\243",                          /*cup (union)*/
"\001\243|\203\351_\311\203|\243",                  /*cap (intersection)*/
"\001\243(\203\302-\345-\303",                      /*subset of*/
"\001\302-\345-\303\203)\243",                      /*superset of*/
"\001_\b\243(\203\302-\345-\303",                   /*improper subset*/
"\001_\b\302-\345-\303\203)\243",                   /*improper superset*/
"\001\016~\017",                                    /*infinity*/
"\001\200o\201\301`\241\341`\241\341`\201\301",     /*partial derivative*/
"\001\016:\017",                                    /*gradient*/
"\001\200-\202\341,\301\242",                       /*not*/
"\001\016?\017",                                    /*integral sign*/
"\002o\242c\202",                                   /*proportional to*/
"\001O\b/",                                         /*empty set*/
"\001<\b\341-\302",                                 /*member of*/
"\001+",                                            /*equation plus*/
"\003(R)",                                          /*registered*/
"\003(C)",                                          /*copyright*/
"\001|",                                            /*box rule */
"\001\033Y",                                        /*cent sign*/
"\001|\b\342=\302",                                 /*double dagger*/
"\002=>",                                           /*right hand*/
"\002<=",                                           /*left hand*/
"\001*",                                            /*math * */
"\001\0162\017",                                    /*\(bs yields small sigma*/
"\001|",                                            /*or (was star)*/
"\001O",                                            /*circle*/
"\001|",                                            /*left top of big brace*/
"\001|",                                            /*left bot of big brace*/
"\001|",                                            /*right top of big brace*/
"\001|",                                            /*right bot of big brace*/
"\001\016]\017",                                    /*left center of big brace*/
"\001\016\\\017",                                   /*right center of big brace*/
"\001|",                                            /*bold vertical*/
"\001|",                                            /*left floor (lb of big bracket)*/
"\001|",                                            /*right floor (rb of big bracket)*/
"\001|",                                            /*left ceiling (lt of big bracket)*/
"\001|"                                             /*right ceiling (rt of big bracket)*/
```

FILES

/usr/lib/term/tab*name*	driving tables
/usr/lib/term/README	list of terminals supported by **nroff**(1)

SEE ALSO

nroff(1), **ttcompat**(4M)

NAME

term – format of compiled term file

SYNOPSIS

term

DESCRIPTION

Compiled **terminfo** descriptions are placed under the directory **/usr/share/lib/terminfo**. In order to avoid a linear search of a huge system directory, a two-level scheme is used: **/usr/share/lib/terminfo/c/name** where *name* is the name of the terminal, and *c* is the first character of *name*. Thus, *act4* can be found in the file **/usr/share/lib/terminfo/a/act4**. Synonyms for the same terminal are implemented by multiple links to the same compiled file.

The format has been chosen so that it will be the same on all hardware. An 8 or more bit byte is assumed, but no assumptions about byte ordering or sign extension are made.

The compiled file is created with the **tic**(8V) program, and read by the routine **setupterm** (see **curses**(3V)). Both of these pieces of software are part of **curses**(3V). The file is divided into six parts:

the header,
terminal names,
boolean flags,
numbers,
strings,
and
string table.

The header section begins the file. This section contains six short integers in the format described below. These integers are:

(1) the magic number (octal 0432);
(2) the size, in bytes, of the names section;
(3) the number of bytes in the boolean section;
(4) the number of short integers in the numbers section;
(5) the number of offsets (short integers) in the strings section;
(6) the size, in bytes, of the string table.

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, other negative value are illegal. The –1 generally means that a capability is missing from this terminal. Note: this format corresponds to the hardware of the VAX and PDP-11. Machines where this does not correspond to the hardware read the integers as two bytes and compute the result.

The terminal names section comes next. It contains the first line of the terminfo description, listing the various names for the terminal, separated by the '|' character. The section is terminated with an ASCII NUL character.

The boolean flags have one byte for each flag. This byte is either 0 or 1 as the flag is present or absent. The capabilities are in the same order as the file **<term.h>**.

Between the boolean section and the number section, a null byte will be inserted, if necessary, to ensure that the number section begins on an even byte. All short integers are aligned on a short word boundary.

The numbers section is similar to the flags section. Each capability takes up two bytes, and is stored as a short integer. If the value represented is –1, 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 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: 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, an octal dump of the description for the Microterm ACT 4 is included:

```
microterm|act4|microterm act iv,
cr=^M, cudl=^J, ind=^J, bel=^G, am, cubl=^H,
ed=^_, el=^^, clear=^L, cup=^T%p1%c%p2%c,
cols#80, lines#24, cufl=^X, cuul=^Z, home=^],

000 032 001      \0 025  \0  \b  \0 212  \0   "  \0   m   i   c   r
020   o   t   e   r   m   |   a   c   t   4   |   m   i   c   r   o
040   t   e   r   m       a   c   t       i   v  \0  \0 001  \0  \0
060  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
100  \0  \0   P  \0 377 377 030  \0 377 377 377 377 377 377 377 377
120 377 377 377 377  \0  \0 002  \0 377 377 377 377 004  \0 006  \0
140  \b  \0 377 377 377 377  \n  \0 026  \0 030  \0 377 377 032  \0
160 377 377 377 377 034  \0 377 377 036  \0 377 377 377 377 377 377
200 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
520 377 377 377 377      \0 377 377 377 377 377 377 377 377 377 377
540 377 377 377 377 377 377 007  \0  \r  \0  \f  \0 036  \0 037  \0
560 024   %   p   1   %   c   %   p   2   %   c  \0  \n  \0 035  \0
600  \b  \0 030  \0 032  \0  \n  \0
```

Some limitations: total compiled entries cannot exceed 4096 bytes. The name field cannot exceed 128 bytes.

FILES

/usr/share/lib/terminfo/*/*
compiled terminal capability data base

SEE ALSO

curses(3V), **terminfo**(5V), **tic**(8V)

NAME

termcap – terminal capability data base

DESCRIPTION

termcap is a data base describing the capabilities of terminals. Terminals are described in **termcap** source descriptions by giving a set of capabilities which they have, by describing how operations are performed, by describing padding requirements, and by specifying initialization sequences. This database is used by applications programs such as vi(1), and libraries such as **curses**(3V), so they can work with a variety of terminals without changes to the programs.

Each **termcap** entry consist of a number of colon-separated (:) fields. The first field for each terminal lists the various names by which it is known, separated by bar (|) characters. The first name is always two characters long, and is used by older (version 6) systems (which store the terminal type in a 16-bit word in a system-wide database). The second name given is the most common abbreviation for the terminal (this is the one to which the environment variable **TERM** would normally be set). The last name should fully identify the terminal's make and model. All other names are taken as synonyms for the initial terminal name. All names but the first and last should be in lower case and contain no blanks; the last name may well contain upper case and blanks for added readability.

Terminal names (except for the last, verbose entry) should be chosen using the following conventions:

- The particular piece of hardware making up the terminal should have a root name chosen; for example, for the Hewlett-Packard 2621, **hp2621**. This name should not contain hyphens.
- Modes that the hardware can be in or user preferences should be indicated by appending a hyphen and an indicator of the mode. Thus, a **vt100** in 132-column mode would be given as: **vt100–w**. The following suffixes should be used where possible:

Suffix	*Meaning*	*Example*
–w	wide mode (more than 80 columns)	**vt100–w**
–am	with automatic margins (usually default)	**vt100–am**
–nam	without automatic margins	**vt100–nam**
–n	number of lines on the screen	**aaa–60**
–na	no arrow keys (leave them in local)	**concept100–na**
*–n***p**	number of pages of memory	**concept100–4p**
–rv	reverse video	**concept100–rv**

Terminal entries may continue onto multiple lines by giving a \ as the last character of a line, and empty fields may be included for readability (here between the last field on a line and the first field on the next). Comments may be included on lines beginning with #.

Types of Capabilities

Terminal capabilities each have a two-letter code, and are of three types:

boolean These indicate particular features of the terminal. For instance, an entry for a terminal that has automatic margins (an automatic RETURN and LINEFEED when the end of a line is reached) would contain a field with the boolean capability **am**.

numeric These give the size of the display of some other attribute. Numeric capabilities are followed by the character '#', and a number. An entry for a teminal with an 80-column display would have a field containing **co#80**.

string These indicate the character sequences used to perform particular terminal operations. String-valued capabilities, such as **ce** (clear-to-end-of-line sequence) are given by the two-letter code, followed by the character '=', and a string (which ends at the following : field delimiter).

A delay factor, in milliseconds may appear after the '='. Padding characters are supplied by **tputs** after the remainder of the string is sent. The delay can be either a number, or a number followed by the character '*', which indicates that the proportional padding is required, in which case the number given is the

amount of padding for each line affected by an operation using that capability. (In the case of an insert-character operation, the factor is still the number of *lines* affected; this is always 1 unless the terminal 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 line to tenths of milliseconds. (Only one decimal place is allowed.)

Comments

To comment-out a capability field, insert a '.' (period) as the first character in that field (following the :).

Escape Sequence Codes

A number of escape sequences are provided in the string-valued capabilities for easy encoding of characters there:

\E	maps to ESC
^X	maps to CTRL-*X* for any appropriate character *X*
\n	maps to LINEFEED
\r	maps to RETURN
\t	maps to TAB
\b	maps to BACKSPACE
\f	maps to FORMFEED

Finally, characters may be given as three octal digits after a backslash (for example, **\123**), and the characters ^ (caret) and \ (backslash) may be given as \^ and \\ respectively.

If it is necessary to place a : in a capability it must be escaped in octal as **\072**.

If it is necessary to place a NUL character in a string capability it must be encoded as **\200**. (The routines that deal with **termcap** use C strings and strip the high bits of the output very late, so that a **\200** comes out as a **\000** would.)

Parameterized Strings

Cursor addressing and other strings requiring parameters are described by a parameterized string capability, with **printf**(3V)-like escapes (*%x*) in it; other characters are passed through unchanged. For example, to address the cursor, the **cm** capability is given, using two parameters: the row and column to move 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 an analogous CM capability.)

The % escapes have the following meanings:

%%	produce the character %
%d	output *value* as in **printf %d**
%2	output *value* as in **printf %2d**
%3	output *value* as in **printf %3d**
%.	output *value* as in **printf %c**
%+x	add *x* to *value*, then do '**%.**'
%>xy	if *value* > *x* then add *y*, no output
%r	reverse order of two parameters, no output
%i	increment by one, no output
%n	exclusive-or all parameters with 0140 (Datamedia 2500)
%B	BCD (16*(*value*/10)) + (*value*%10), no output
%D	Reverse coding (*value* – 2*(*value*%16)), no output (Delta Data)

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: the order of the row and column coordinates is reversed here and that the row and column are sent as two-digit integers. Thus its **cm** capability is **':cm=6\E&%r%2c%2Y:'**. Terminals that use **'%.'** need to be able to backspace the cursor (**le**) and to move the cursor up one line on the screen (**up**). This is necessary because it is not always safe to transmit **\n**, **^D**, and **\r**, as the system may change or discard them. (Programs using **termcap** must set terminal modes so that TAB characters are not expanded, making **\t** safe to send. This turns out to be essential for the Ann Arbor 4080.)

A final example is the Lear Siegler ADM–3a, which offsets row and column by a blank character, thus it requires **':cm=\E=%+ %+:'**.

Row or column absolute cursor addressing can be given as single-parameter capabilities **ch** (horizontal position absolute) and **cv** (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 **cm**. If there are parameterized local motions (for example, move *n* positions to the right) these can be given as **DO**, **LE**, **RI**, and **UP** with a single parameter indicating how many positions to move. These are primarily useful if the terminal does not have **cm**, such as the Tektronix 4025.

Delays

Certain capabilities control padding in the terminal driver. These are primarily needed by hardcopy terminals and are used by the **tset** (1) program to set terminal driver modes appropriately. Delays embedded in the capabilities **cr**, **sf**, **le**, **ff**, and **ta** will set the appropriate delay bits in the terminal driver. If **pb** (padding baud rate) is given, these values can be ignored at baud rates below the value of **pb**. For 4.2BSD **tset**, the delays are given as numeric capabilities **dC**, **dN**, **dB**, **dF**, and **dT** instead.

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 **tc** can be given with the name of the similar terminal. This capability must be *last*, and the combined length of the entries must not exceed 1024. The capabilities given before **tc** override those in the terminal type invoked by **tc**. A capability can be canceled by placing **xx@** to the left of the **tc** invocation, where *xx* is the capability. For example, the entry

hn|2621–nl:ks@:ke@:tc=2621:

defines a **2621–nl** that does not have the **ks** or **ke** capabilities, hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

CAPABILITIES

The characters in the *Notes* field in the next table have the following meanings (more than one may apply to a capability):

- **N** indicates numeric parameter(s)
- **P** indicates that padding may be specified
- ***** indicates that padding may be based on the number of lines affected
- **o** indicates capability is obsolete

Obsolete capabilities have no **terminfo** equivalents, since they were considered useless, or are subsumed by other capabilities. New software should not rely on them.

Name	*Type*	*Notes*	*Description*
!1	*str*		sent by shifted save key
!2	*str*		sent by shifted suspend key
!3	*str*		sent by shifted undo key
#1	*str*		sent by shifted help key
#2	*str*		sent by shifted home key
#3	*str*		sent by shifted input key
#4	*str*		sent by shifted left-arrow key
%0	*str*		sent by redo key
%1	*str*		sent by help key

%2	*str*		sent by mark key
%3	*str*		sent by message key
%4	*str*		sent by move key
%5	*str*		sent by next-object key
%6	*str*		sent by open key
%7	*str*		sent by options key
%8	*str*		sent by previous-object key
%9	*str*		sent by print or copy key
%a	*str*		sent by shifted message key
%b	*str*		sent by shifted move key
%c	*str*		sent by shifted next-object key
%d	*str*		sent by shifted options key
%e	*str*		sent by shifted previous-object key
%f	*str*		sent by shifted print or copy key
%g	*str*		sent by shifted redo key
%h	*str*		sent by shifted replace key
%i	*str*		sent by shifted right-arrow key
%j	*str*		sent by shifted resume key
&0	*str*		sent by shifted cancel key
&1	*str*		sent by ref(erence) key
&2	*str*		sent by refresh key
&3	*str*		sent by replace key
&4	*str*		sent by restart key
&5	*str*		sent by resume key
&6	*str*		sent by save key
&7	*str*		sent by suspend key
&8	*str*		sent by undo key
&9	*str*		sent by shifted beg(inning) key
***0**	*str*		sent by shifted find key
***1**	*str*		sent by shifted cmd (command) key
***2**	*str*		sent by shifted copy key
***3**	*str*		sent by shifted create key
***4**	*str*		sent by shifted delete-char key
***5**	*str*		sent by shifted delete-line key
***6**	*str*		sent by select key
***7**	*str*		sent by shifted end key
***8**	*str*		sent by shifted clear-line key
***9**	*str*		sent by shifted exit key
5i	*bool*		printer will not echo on screen
@0	*str*		sent by find key
@1	*str*		sent by beg(inning) key
@2	*str*		sent by cancel key
@3	*str*		sent by close key
@4	*str*		sent by cmd (command) key
@5	*str*		sent by copy key
@6	*str*		sent by create key
@7	*str*		sent by end key
@8	*str*		sent by enter/send key (unreliable)
@9	*str*		sent by exit key
AL	*str*	*(NP*)*	add *n* new blank lines
CC	*str*		terminal settable command character in prototype
CM	*str*	*(NP)*	memory-relative cursor motion to row *m*, column *n*
DC	*str*	*(NP*)*	delete *n* characters
DL	*str*	*(NP*)*	delete *n* lines
DO	*str*	*(NP*)*	move cursor down *n* lines
EP	*bool*	*(o)*	even parity
F1-F9	*str*		sent by function keys 11-19
FA-FZ	*str*		sent by function keys 20-45

Fa-Fr	*str*		sent by function keys 46-63
HC	*bool*		cursor is hard to see
HD	*bool*	*(o)*	half-duplex
IC	*str*	*(NP*)*	insert *n* blank characters
K1	*str*		sent by keypad upper left
K2	*str*		sent by keypad center
K3	*str*		sent by keypad upper right
K4	*str*		sent by keypad lower left
K5	*str*		sent by keypad lower right
LC	*bool*	*(o)*	lower-case only
LE	*str*	*(NP)*	move cursor left *n* positions
LF	*str*	*(P)*	turn off soft labels
LO	*str*	*(P)*	turn on soft labels
MC	*str*	*(P)*	clear left and right soft margins
ML	*str*	*(P)*	set soft left margin
MR	*str*	*(P)*	set soft right margin
NL	*bool*	*(o)*	\n is NEWLINE, not LINEFEED
NP	*bool*		pad character does not exist
NR	*bool*		**ti** does not reverse **te**
Nl	*num*		number of labels on screen (start at 1)
OP	*bool*	*(o)*	odd parity
RA	*str*	*(P)*	turn off automatic margins
RF	*str*		send next input character (for ptys)
RI	*str*	*(NP)*	move cursor right *n* positions
RX	*str*	*(P)*	turn off xoff/xon handshaking
SA	*str*	*(P)*	turn on automatic margins
SF	*str*	*(NP*)*	scroll forward *n* lines
SR	*str*	*(NP*)*	scroll backward *n* lines
SX	*str*	*(P)*	turn on xoff/xon handshaking
UC	*bool*	*(o)*	upper-case only
UP	*str*	*(NP*)*	move cursor up *n* lines
XF	*str*		x-off character (default DC3)
XN	*str*		x-on character (default DC1)
ac	*str*		graphic character set pairs aAbBcC – def=VT100
ae	*str*	*(P)*	end alternate character set
al	*str*	*(P*)*	add new blank line
am	*bool*		terminal has automatic margins
as	*str*	*(P)*	start alternate character set
bc	*str*	*(o)*	backspace if not ^H
bl	*str*	*(P)*	audible signal (bell)
bs	*bool*	*(o)*	terminal can backspace with ^H
bt	*str*	*(P)*	back-tab
bw	*bool*		**le** (backspace) wraps from column 0 to last column
cb	*str*	*(P)*	clear to beginning of line, inclusive
cd	*str*	*(P*)*	clear to end of display
ce	*str*	*(P)*	clear to end of line
ch	*str*	*(NP)*	set cursor column (horizontal position)
cl	*str*	*(P*)*	clear screen and home cursor
cm	*str*	*(NP)*	screen-relative cursor motion to row *m*, column *n*
co	*num*		number of columns in a line
cr	*str*	*(P*)*	RETURN
cs	*str*	*(NP)*	change scrolling region to lines *m* through *n* (VT100)
ct	*str*	*(P)*	clear all tab stops
cv	*str*	*(NP)*	set cursor row (vertical position)
dB	*num*	*(o)*	milliseconds of **bs** delay needed (default 0)
dC	*num*	*(o)*	milliseconds of **cr** delay needed (default 0)
dF	*num*	*(o)*	milliseconds of **ff** delay needed (default 0)
dN	*num*	*(o)*	milliseconds of **nl** delay needed (default 0)

dT	*num*	*(o)*	milliseconds of horizontal tab delay needed (default 0)
dV	*num*	*(o)*	milliseconds of vertical tab delay needed (default 0)
da	*bool*		display may be retained above the screen
db	*bool*		display may be retained below the screen
dc	*str*	*(P*)*	delete character
dl	*str*	*(P*)*	delete line
dm	*str*		enter delete mode
do	*str*		down one line
ds	*str*		disable status line
eA	*str*	*(P)*	enable graphic character set
ec	*str*	*(NP)*	erase *n* characters
ed	*str*		end delete mode
ei	*str*		end insert mode
eo	*bool*		can erase overstrikes with a blank
es	*bool*		escape can be used on the status line
ff	*str*	*(P*)*	hardcopy terminal page eject
fs	*str*		return from status line
gn	*bool*		generic line type (for example dialup, switch)
hc	*bool*		hardcopy terminal
hd	*str*		half-line down (forward 1/2 linefeed)
ho	*str*	*(P)*	home cursor
hs	*bool*		has extra "status line"
hu	*str*		half-line up (reverse 1/2 linefeed)
hz	*bool*		cannot print ~s (Hazeltine)
i1	*str*		terminal initialization string (**terminfo** only)
i3	*str*		terminal initialization string (**terminfo** only)
iP	*str*		pathname of program for initialization (**terminfo** only)
ic	*str*	*(P*)*	insert character
if	*str*		name of file containing initialization string
im	*str*		enter insert mode
in	*bool*		insert mode distinguishes nulls
ip	*str*	*(P*)*	insert pad after character inserted
is	*str*		terminal initialization string
it	*num*		tab stops initially every *n* positions
k0-k9	*str*		sent by function keys 0-9
k;	*str*		sent by function key 10
kA	*str*		sent by insert-line key
kB	*str*		sent by back-tab key
kC	*str*		sent by clear-screen or erase key
kD	*str*		sent by delete-character key
kE	*str*		sent by clear-to-end-of-line key
kF	*str*		sent by scroll-forward/down key
kH	*str*		sent by home-down key
kI	*str*		sent by insert-character or enter-insert-mode key
kL	*str*		sent by delete-line key
kM	*str*		sent by insert key while in insert mode
kN	*str*		sent by next-page key
kP	*str*		sent by previous-page key
kR	*str*		sent by scroll-backward/up key
kS	*str*		sent by clear-to-end-of-screen key
kT	*str*		sent by set-tab key
ka	*str*		sent by clear-all-tabs key
kb	*str*		sent by backspace key
kd	*str*		sent by down-arrow key
ke	*str*		out of "keypad transmit" mode
kh	*str*		sent by home key
kl	*str*		sent by left-arrow key
km	*bool*		has a "meta" key (shift, sets parity bit)

kn	*num*	*(o)*	number of function (**k0**–**k9**) keys (default 0)
ko	*str*	*(o)*	termcap entries for other non-function keys
kr	*str*		sent by right-arrow key
ks	*str*		put terminal in "keypad transmit" mode
kt	*str*		sent by clear-tab key
ku	*str*		sent by up-arrow key
l0-l9	*str*		labels on function keys 0-9 if not f0-f9
la	*str*		label on function key 10 if not f10
le	*str*	*(P)*	move cursor left one position
lh	*num*		number of rows in each label
li	*num*		number of lines on screen or page
ll	*str*		last line, first column
lm	*num*		lines of memory if > **li** (0 means varies)
lw	*num*		number of columns in each label
ma	*str*	*(o)*	arrow key map (used by *vi* version 2 only)
mb	*str*		turn on blinking attribute
md	*str*		turn on bold (extra bright) attribute
me	*str*		turn off all attributes
mh	*str*		turn on half-bright attribute
mi	*bool*		safe to move while in insert mode
mk	*str*		turn on blank attribute (characters invisible)
ml	*str*	*(o)*	memory lock on above cursor
mm	*str*		turn on "meta mode" (8th bit)
mo	*str*		turn off "meta mode"
mp	*str*		turn on protected attribute
mr	*str*		turn on reverse-video attribute
ms	*bool*		safe to move in standout modes
mu	*str*	*(o)*	memory unlock (turn off memory lock)
nc	*bool*	*(o)*	no correctly-working **cr** (Datamedia 2500, Hazeltine 2000)
nd	*str*		non-destructive space (cursor right)
nl	*str*	*(o)*	NEWLINE character if not
ns	*bool*	*(o)*	terminal is a CRT but does not scroll
nw	*str*	*(P)*	NEWLINE (behaves like **cr** followed by **do**)
nx	*bool*		padding will not work, xoff/xon required
os	*bool*		terminal overstrikes
pO	*str*	*(N)*	turn on the printer for *n* bytes
pb	*num*		lowest baud where delays are required
pc	*str*		pad character (default NUL)
pf	*str*		turn off the printer
pk	*str*		program function key *n* to type string *s* (**terminfo** only)
pl	*str*		program function key *n* to execute string *s* (**terminfo** only)
pn	*str*	*(NP)*	program label *n* to show string *s* (**terminfo** only)
po	*str*		turn on the printer
ps	*str*		print contents of the screen
pt	*bool*	*(o)*	has hardware tab stops (may need to be set with **is**)
px	*str*		program function key *n* to transmit string *s* (**terminfo** only)
r1	*str*		reset terminal completely to sane modes (**terminfo** only)
r2	*str*		reset terminal completely to sane modes (**terminfo** only)
r3	*str*		reset terminal completely to sane modes (**terminfo** only)
rP	*str*	*(P)*	like **ip** but when in replace mode
rc	*str*	*(P)*	restore cursor to position of last **sc**
rf	*str*		name of file containing reset string
ri	*?*		unknown at present
rp	*str*	*(NP*)*	repeat character *c* *n* times
rs	*str*		reset terminal completely to sane modes
sa	*str*	*(NP)*	define the video attributes (9 parameters)
sc	*str*	*(P)*	save cursor position
se	*str*		end standout mode

sf	*str*	*(P)*	scroll text up
sg	*num*		number of garbage chars left by **so** or **se** (default 0)
so	*str*		begin standout mode
sr	*str*	*(P)*	scroll text down
st	*str*		set a tab stop in all rows, current column
ta	*str*	*(P)*	move cursor to next 8-position hardware tab stop
tc	*str*		entry of similar terminal – must be last
te	*str*		string to end programs that use **termcap**
ti	*str*		string to begin programs that use **termcap**
ts	*str*	*(N)*	go to status line, column *n*
uc	*str*		underscore one character and move past it
ue	*str*		end underscore mode
ug	*num*		number of garbage chars left by **us** or **ue** (default 0)
ul	*bool*		underline character overstrikes
up	*str*		upline (cursor up)
us	*str*		start underscore mode
vb	*str*		visible bell (must not move cursor)
ve	*str*		make cursor appear normal (undo **vs**/**vi**)
vi	*str*		make cursor invisible
vs	*str*		make cursor very visible
vt	*num*		virtual terminal number (not supported on all systems)
wi	*str*	*(N)*	set current window to lines *i* through *j*, columns *m* through *n*
ws	*num*		number of columns in status line
xb	*bool*		Beehive (f1=ESC, f2=^C)
xn	*bool*		NEWLINE ignored after 80 cols (Concept)
xo	*bool*		terminal uses xoff/xon handshaking
xr	*bool*	*(o)*	RETURN acts like **ce cr nl** (Delta Data)
xs	*bool*		standout not erased by overwriting (Hewlett-Packard)
xt	*bool*		TAB characters destructive, magic **so** char (Teleray 1061)
xx	*bool*	*(o)*	Tektronix 4025 insert-line

ENVIRONMENT

If the environment variable **TERMCAP** contains an absolute pathname, programs look to that file for terminal descriptions, rather than **/usr/share/lib/termcap**. If the value of this variable is in the form of a **termcap** entry, programs use that value for the terminal description.

FILES

/usr/share/lib/termcap file containing terminal descriptions

SEE ALSO

ex(1), **more**(1), **tset**(1), **ul**(1), **vi**(1), **curses**(3V), **printf**(3V), **termcap**(3X), **term**(5V), **terminfo**(5V)

System and Network Administration

WARNINGS

UNIX System V uses **terminfo**(5V) rather than **termcap**. SunOS supports either **termcap** or **terminfo**(5V) terminal databases, depending on whether you link with the **termcap**(3X) or **curses**(3V) libraries. Transitions between the two should be relatively painless if capabilities flagged as "obsolete" are avoided.

vi allows only 256 characters for string capabilities, and the routines in **termcap**(3X) do not check for overflow of this buffer. The total length of a single entry (excluding only escaped NEWLINE characters) may not exceed 1024.

Not all programs support all entries.

NAME

terminfo – terminal capability data base

SYNOPSIS

/usr/share/lib/terminfo/?/*

AVAILABILITY

This database is available with the *System V* software installation option. Refer to *Installing SunOS 4.1* for information on how to install optional software.

DESCRIPTION

terminfo is a compiled database (see **tic**(8V)) describing the capabilities of terminals. Terminals are described in **terminfo** source descriptions by giving a set of capabilities which they have, by describing how operations are performed, by describing padding requirements, and by specifying initialization sequences. This database is used by applications programs, and by libraries such as **curses**(3V), so they can work with a variety of terminals without changes to the programs. To obtain the source description for a terminal, use the **–I** option of **infocmp**(8V).

Entries in **terminfo** source files consist of a number of comma-separated fields. White space after each comma is ignored. The first line of each terminal description in the **terminfo** database gives the name by which **terminfo** knows the terminal, separated by pipe (|) characters. The first name given is the most common abbreviation for the terminal (this is the one to which the environment variable **TERM** would normally be set), the last name given should be a long name fully identifying the terminal, and all others are understood as synonyms for the terminal name. All names but the last should contain no blanks; the last name may contain blanks for readability.

Terminal names (except for the last, verbose entry) should be chosen using the following conventions:

- The particular piece of hardware making up the terminal should have a root name chosen; for example, for the Hewlett-Packard 2621, **hp2621**. This name should not contain hyphens.
- Modes that the hardware can be in or user preferences should be indicated by appending a hyphen and an indicator of the mode. Thus, a **vt100** in 132-column mode would be given as: **vt100–w**. The following suffixes should be used where possible:

Suffix	*Meaning*	*Example*
–w	wide mode (more than 80 columns)	**vt100–w**
–am	with automatic margins (usually default)	**vt100–am**
–nam	without automatic margins	**vt100–nam**
–n	number of lines on the screen	**aaa–60**
–na	no arrow keys (leave them in local)	**concept100–na**
*–n***p**	number of pages of memory	**concept100–4p**
–rv	reverse video	**concept100–rv**

CAPABILITIES

In the table below, the **Variable** is the name by which the C programmer (at the **terminfo** level) accesses the capability. The **capname** is the short name for this variable used in the text of the database. It is used by a person updating the database and by the **tput**(1V) command when asking what the value of the capability is for a particular terminal. The **Termcap Code** is a two-letter code that corresponds to the old **termcap** capability name.

Capability names have no hard length limit, but an informal limit of 5 characters has been adopted to keep them short. Whenever possible, names are chosen to be the same as or similar to the ANSI X3.64-1979 standard. Semantics are also intended to match those of the specification.

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 table below, have names beginning with '**key_**'. The following indicators may appear at the end of the **Description** for a variable.

(G) indicates that the string is passed through **tparm**( ) with parameters (parms) as given ($\#_i$).
(*) indicates that padding may be based on the number of lines affected.
($\#_i$) indicates the ith parameter.

Variable	*Capname*	*Termcap*	*Description*
Boolean			
auto_left_margin	**bw**	**bw**	**cub**1 wraps from column 0 to last column
auto_right_margin	**am**	**am**	Terminal has automatic margins
no_esc_ctlc	**xsb**	**xb**	Beehive (f1=ESC, f2=^C)
ceol_standout_glitch	**xhp**	**xs**	Standout not erased by overwriting (Hewlett-Packard)
eat_newline_glitch	**xenl**	**xn**	NEWLINE ignored after 80 cols (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_status_line	**hs**	**hs**	Has extra "status line"
insert_null_glitch	**in**	**in**	Insert mode distinguishes nulls
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 will not work, xon/xoff required
non_rev_rmcup	**nrrmc**	**NR**	**smcup** does not reverse **rmcup**
no_pad_char	**npc**	**NP**	Pad character does not exist
over_strike	**os**	**os**	Terminal overstrikes on hard-copy terminal
prtr_silent	**mc5i**	**5i**	Printer will not echo on screen
status_line_esc_ok	**eslok**	**es**	Escape can be used on the status line
dest_tabs_magic_smso	**xt**	**xt**	Destructive TAB characters, magic **smso** char (Teleray 1061)
tilde_glitch	**hz**	**hz**	Hazeltine; cannot print tildes(~)
transparent_underline	**ul**	**ul**	Underline character overstrikes
xon_xoff	**xon**	**xo**	Terminal uses xon/xoff handshaking
Number			
columns	**cols**	**co**	Number of columns in a line
init_tabs	**it**	**it**	tab stops initially every # spaces
label_height	**lh**	**lh**	Number of rows in each label
label_width	**lw**	**lw**	Number of cols in each label
lines	**lines**	**li**	Number of lines on screen or page
lines_of_memory	**lm**	**lm**	Lines of memory if > **lines**; **0** means varies
magic_cookie_glitch	**xmc**	**sg**	Number blank chars left by **smso** or **rmso**
num_labels	**nlab**	**Nl**	Number of labels on screen (start at 1)
padding_baud_rate	**pb**	**pb**	Lowest baud rate where padding needed
virtual_terminal	**vt**	**vt**	Virtual terminal number (not supported on all systems)
width_status_line	**wsl**	**ws**	Number of columns in status line
String			
acs_chars	**acsc**	**ac**	Graphic charset pairs aAbBcC - def=VT100
back_tab	**cbt**	**bt**	Back tab
bell	**bel**	**bl**	Audible signal (bell)
carriage_return	**cr**	**cr**	RETURN (*)
change_scroll_region	**csr**	**cs**	Change to lines #1 through #2 (VT100) (G)

char_padding	**rmp**	**rP**	Like **ip** but when in replace mode
clear_all_tabs	**tbc**	**ct**	Clear all tab stops
clear_margins	**mgc**	**MC**	Clear left and right soft margins
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 (*)
column_address	**hpa**	**ch**	Horizontal position absolute (G)
command_character	**cmdch**	**CC**	Terminal settable command char in prototype
cursor_address	**cup**	**cm**	Cursor motion to row #1 col #2 (G)
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 cursor left one SPACE
cursor_mem_address	**mrcup**	**CM**	Memory relative cursor addressing (G)
cursor_normal	**cnorm**	**ve**	Make cursor appear normal (undo **cvvis/civis**)
cursor_right	**cuf1**	**nd**	Non-destructive space (cursor 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
delete_character	**dch1**	**dc**	Delete character (*)
delete_line	**dl1**	**dl**	Delete line (*)
dis_status_line	**dsl**	**ds**	Disable status line
down_half_line	**hd**	**hd**	Half-line down (forward 1/2 LINEFEED)
ena_acs	**enacs**	**eA**	Enable alternate char set
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_insert_mode	**smir**	**im**	Insert mode (enter);
enter_protected_mode	**prot**	**mp**	Turn on protected mode
enter_reverse_mode	**rev**	**mr**	Turn on reverse video mode
enter_secure_mode	**invis**	**mk**	Turn on blank mode (chars invisible)
enter_standout_mode	**smso**	**so**	Begin standout mode
enter_underline_mode	**smul**	**us**	Start underscore mode
enter_xon_mode	**smxon**	**SX**	Turn on xon/xoff handshaking
erase_chars	**ech**	**ec**	Erase #1 characters (G)
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_insert_mode	**rmir**	**ei**	End insert mode;
exit_standout_mode	**rmso**	**se**	End standout mode
exit_underline_mode	**rmul**	**ue**	End underscore mode
exit_xon_mode	**rmxon**	**RX**	Turn off xon/xoff handshaking
flash_screen	**flash**	**vb**	Visible bell (must not move cursor)
form_feed	**ff**	**ff**	Hardcopy terminal page eject (*)
from_status_line	**fsl**	**fs**	Return from status line
init_1string	**is1**	**i1**	Terminal initialization string
init_2string	**is2**	**is**	Terminal initialization string
init_3string	**is3**	**i3**	Terminal initialization string
init_file	**if**	**if**	Name of initialization file containing **is**
init_prog	**iprog**	**iP**	Path name of program for init
insert_character	**ich1**	**ic**	Insert character

insert_line	**il1**	**al**	Add new blank line (*)
insert_padding	**ip**	**ip**	Insert pad after character inserted (*)
key_a1	**ka1**	**K1**	KEY_A1, 0534, Upper left of keypad
key_a3	**ka3**	**K3**	KEY_A3, 0535, Upper right of keypad
key_b2	**kb2**	**K2**	KEY_B2, 0536, Center of keypad
key_backspace	**kbs**	**kb**	KEY_BACKSPACE, 0407, Sent by BACKSPACE key
key_beg	**kbeg**	**@1**	KEY_BEG, 0542, Sent by beg(inning) key
key_btab	**kcbt**	**kB**	KEY_BTAB, 0541, Sent by back-tab key
key_c1	**kc1**	**K4**	KEY_C1, 0537, Lower left of keypad
key_c3	**kc3**	**K5**	KEY_C3, 0540, Lower right of keypad
key_cancel	**kcan**	**@2**	KEY_CANCEL, 0543, Sent by cancel key
key_catab	**ktbc**	**ka**	KEY_CATAB, 0526, Sent by clear-all-tabs key
key_clear	**kclr**	**kC**	KEY_CLEAR, 0515, Sent by clear- screen or erase key
key_close	**kclo**	**@3**	KEY_CLOSE, 0544, Sent by close key
key_command	**kcmd**	**@4**	KEY_COMMAND, 0545, Sent by cmd (command) key
key_copy	**kcpy**	**@5**	KEY_COPY, 0546, Sent by copy key
key_create	**kcrt**	**@6**	KEY_CREATE, 0547, Sent by create key
key_ctab	**kctab**	**kt**	KEY_CTAB, 0525, Sent by clear-tab key
key_dc	**kdch1**	**kD**	KEY_DC, 0512, Sent by delete-character key
key_dl	**kdl1**	**kL**	KEY_DL, 0510, Sent by delete-line key
key_down	**kcud1**	**kd**	KEY_DOWN, 0402, Sent by terminal down-arrow key
key_eic	**krmir**	**kM**	KEY_EIC, 0514, Sent by **rmir** or **smir** in insert mode
key_end	**kend**	**@7**	KEY_END, 0550, Sent by end key
key_enter	**kent**	**@8**	KEY_ENTER, 0527, Sent by enter/send key
key_eol	**kel**	**kE**	KEY_EOL, 0517, Sent by clear-to-end- of-line key
key_eos	**ked**	**kS**	KEY_EOS, 0516, Sent by clear-to-end- of-screen key
key_exit	**kext**	**@9**	KEY_EXIT, 0551, Sent by exit key
key_f0	**kf0**	**k0**	KEY_F(0), 0410, Sent by function key f0
key_f1	**kf1**	**k1**	KEY_F(1), 0411, Sent by function key f1
key_f2	**kf2**	**k2**	KEY_F(2), 0412, Sent by function key f2
key_f3	**kf3**	**k3**	KEY_F(3), 0413, Sent by function key f3
key_f4	**kf4**	**k4**	KEY_F(4), 0414, Sent by function key f4
key_f5	**kf5**	**k5**	KEY_F(5), 0415, Sent by function key f5
key_f6	**kf6**	**k6**	KEY_F(6), 0416, Sent by function key f6
key_f7	**kf7**	**k7**	KEY_F(7), 0417, Sent by function key f7
key_f8	**kf8**	**k8**	KEY_F(8), 0420, Sent by function key f8
key_f9	**kf9**	**k9**	KEY_F(9), 0421, Sent by function key f9
key_f10	**kf10**	**k;**	KEY_F(10), 0422, Sent by function key f10
key_f11	**kf11**	**F1**	KEY_F(11), 0423, Sent by function key f11
key_f12	**kf12**	**F2**	KEY_F(12), 0424, Sent by function key f12
key_f13	**kf13**	**F3**	KEY_F(13), 0425, Sent by function key f13
key_f14	**kf14**	**F4**	KEY_F(14), 0426, Sent by function key f14
key_f15	**kf15**	**F5**	KEY_F(15), 0427, Sent by function key f15
key_f16	**kf16**	**F6**	KEY_F(16), 0430, Sent by function key f16
key_f17	**kf17**	**F7**	KEY_F(17), 0431, Sent by function key f17
key_f18	**kf18**	**F8**	KEY_F(18), 0432, Sent by function key f18
key_f19	**kf19**	**F9**	KEY_F(19), 0433, Sent by function key f19
key_f20	**kf20**	**FA**	KEY_F(20), 0434, Sent by function key f20
key_f21	**kf21**	**FB**	KEY_F(21), 0435, Sent by function key f21
key_f22	**kf22**	**FC**	KEY_F(22), 0436, Sent by function key f22
key_f23	**kf23**	**FD**	KEY_F(23), 0437, Sent by function key f23
key_f24	**kf24**	**FE**	KEY_F(24), 0440, Sent by function key f24
key_f25	**kf25**	**FF**	KEY_F(25), 0441, Sent by function key f25
key_f26	**kf26**	**FG**	KEY_F(26), 0442, Sent by function key f26
key_f27	**kf27**	**FH**	KEY_F(27), 0443, Sent by function key f27
key_f28	**kf28**	**FI**	KEY_F(28), 0444, Sent by function key f28
key_f29	**kf29**	**FJ**	KEY_F(29), 0445, Sent by function key f29
key_f30	**kf30**	**FK**	KEY_F(30), 0446, Sent by function key f30

key_f31	**kf31**	**FL**	KEY_F(31), 0447, Sent by function key f31
key_f32	**kf32**	**FM**	KEY_F(32), 0450, Sent by function key f32
key_f33	**kf33**	**FN**	KEY_F(13), 0451, Sent by function key f13
key_f34	**kf34**	**FO**	KEY_F(34), 0452, Sent by function key f34
key_f35	**kf35**	**FP**	KEY_F(35), 0453, Sent by function key f35
key_f36	**kf36**	**FQ**	KEY_F(36), 0454, Sent by function key f36
key_f37	**kf37**	**FR**	KEY_F(37), 0455, Sent by function key f37
key_f38	**kf38**	**FS**	KEY_F(38), 0456, Sent by function key f38
key_f39	**kf39**	**FT**	KEY_F(39), 0457, Sent by function key f39
key_f40	**kf40**	**FU**	KEY_F(40), 0460, Sent by function key f40
key_f41	**kf41**	**FV**	KEY_F(41), 0461, Sent by function key f41
key_f42	**kf42**	**FW**	KEY_F(42), 0462, Sent by function key f42
key_f43	**kf43**	**FX**	KEY_F(43), 0463, Sent by function key f43
key_f44	**kf44**	**FY**	KEY_F(44), 0464, Sent by function key f44
key_f45	**kf45**	**FZ**	KEY_F(45), 0465, Sent by function key f45
key_f46	**kf46**	**Fa**	KEY_F(46), 0466, Sent by function key f46
key_f47	**kf47**	**Fb**	KEY_F(47), 0467, Sent by function key f47
key_f48	**kf48**	**Fc**	KEY_F(48), 0470, Sent by function key f48
key_f49	**kf49**	**Fd**	KEY_F(49), 0471, Sent by function key f49
key_f50	**kf50**	**Fe**	KEY_F(50), 0472, Sent by function key f50
key_f51	**kf51**	**Ff**	KEY_F(51), 0473, Sent by function key f51
key_f52	**kf52**	**Fg**	KEY_F(52), 0474, Sent by function key f52
key_f53	**kf53**	**Fh**	KEY_F(53), 0475, Sent by function key f53
key_f54	**kf54**	**Fi**	KEY_F(54), 0476, Sent by function key f54
key_f55	**kf55**	**Fj**	KEY_F(55), 0477, Sent by function key f55
key_f56	**kf56**	**Fk**	KEY_F(56), 0500, Sent by function key f56
key_f57	**kf57**	**Fl**	KEY_F(57), 0501, Sent by function key f57
key_f58	**kf58**	**Fm**	KEY_F(58), 0502, Sent by function key f58
key_f59	**kf59**	**Fn**	KEY_F(59), 0503, Sent by function key f59
key_f60	**kf60**	**Fo**	KEY_F(60), 0504, Sent by function key f60
key_f61	**kf61**	**Fp**	KEY_F(61), 0505, Sent by function key f61
key_f62	**kf62**	**Fq**	KEY_F(62), 0506, Sent by function key f62
key_f63	**kf63**	**Fr**	KEY_F(63), 0507, Sent by function key f63
key_find	**kfnd**	**@0**	KEY_FIND, 0552, Sent by find key
key_help	**khlp**	**%1**	KEY_HELP, 0553, Sent by help key
key_home	**khome**	**kh**	KEY_HOME, 0406, Sent by home key
key_ic	**kich1**	**kI**	KEY_IC, 0513, Sent by ins-char/enter ins-mode key
key_il	**kil1**	**kA**	KEY_IL, 0511, Sent by insert-line key
key_left	**kcub1**	**kl**	KEY_LEFT, 0404, Sent by terminal left-arrow key
key_ll	**kll**	**kH**	KEY_LL, 0533, Sent by home-down key
key_mark	**kmrk**	**%2**	KEY_MARK, 0554, Sent by mark key
key_message	**kmsg**	**%3**	KEY_MESSAGE, 0555, Sent by message key
key_move	**kmov**	**%4**	KEY_MOVE, 0556, Sent by move key
key_next	**knxt**	**%5**	KEY_NEXT, 0557, Sent by next-object key
key_npage	**knp**	**kN**	KEY_NPAGE, 0522, Sent by next-page key
key_open	**kopn**	**%6**	KEY_OPEN, 0560, Sent by open key
key_options	**kopt**	**%7**	KEY_OPTIONS, 0561, Sent by options key
key_ppage	**kpp**	**kP**	KEY_PPAGE, 0523, Sent by previous-page key
key_previous	**kprv**	**%8**	KEY_PREVIOUS, 0562, Sent by previous-object key
key_print	**kprt**	**%9**	KEY_PRINT, 0532, Sent by print or copy key
key_redo	**krdo**	**%0**	KEY_REDO, 0563, Sent by redo key
key_reference	**kref**	**&1**	KEY_REFERENCE, 0564, Sent by ref(erence) key
key_refresh	**krfr**	**&2**	KEY_REFRESH, 0565, Sent by refresh key
key_replace	**krpl**	**&3**	KEY_REPLACE, 0566, Sent by replace key
key_restart	**krst**	**&4**	KEY_RESTART, 0567, Sent by restart key
key_resume	**kres**	**&5**	KEY_RESUME, 0570, Sent by resume key
key_right	**kcuf1**	**kr**	KEY_RIGHT, 0405, Sent by terminal right-arrow key
key_save	**ksav**	**&6**	KEY_SAVE, 0571, Sent by save key

key_sbeg	**kBEG**	**&9**	KEY_SBEG, 0572, Sent by shifted beginning key
key_scancel	**kCAN**	**&0**	KEY_SCANCEL, 0573, Sent by shifted cancel key
key_scommand	**kCMD**	***1**	KEY_SCOMMAND, 0574, Sent by shifted command key
key_scopy	**kCPY**	***2**	KEY_SCOPY, 0575, Sent by shifted copy key
key_screate	**kCRT**	***3**	KEY_SCREATE, 0576, Sent by shifted create key
key_sdc	**kDC**	***4**	KEY_SDC, 0577, Sent by shifted delete-char key
key_sdl	**kDL**	***5**	KEY_SDL, 0600, Sent by shifted delete-line key
key_select	**kslt**	***6**	KEY_SELECT, 0601, Sent by select key
key_send	**kEND**	***7**	KEY_SEND, 0602, Sent by shifted end key
key_seol	**kEOL**	***8**	KEY_SEOL, 0603, Sent by shifted clear-line key
key_sexit	**kEXT**	***9**	KEY_SEXIT, 0604, Sent by shifted exit key
key_sf	**kind**	**kF**	KEY_SF, 0520, Sent by scroll-forward/down key
key_sfind	**kFND**	***0**	KEY_SFIND, 0605, Sent by shifted find key
key_shelp	**kHLP**	**#1**	KEY_SHELP, 0606, Sent by shifted help key
key_shome	**kHOM**	**#2**	KEY_SHOME, 0607, Sent by shifted home key
key_sic	**kIC**	**#3**	KEY_SIC, 0610, Sent by shifted input key
key_sleft	**kLFT**	**#4**	KEY_SLEFT, 0611, Sent by shifted left-arrow key
key_smessage	**kMSG**	**%a**	KEY_SMESSAGE, 0612, Sent by shifted message key
key_smove	**kMOV**	**%b**	KEY_SMOVE, 0613, Sent by shifted move key
key_snext	**kNXT**	**%c**	KEY_SNEXT, 0614, Sent by shifted next key
key_soptions	**kOPT**	**%d**	KEY_SOPTIONS, 0615, Sent by shifted options key
key_sprevious	**kPRV**	**%e**	KEY_SPREVIOUS, 0616, Sent by shifted prev key
key_sprint	**kPRT**	**%f**	KEY_SPRINT, 0617, Sent by shifted print key
key_sr	**kri**	**kR**	KEY_SR, 0521, Sent by scroll-backward/up key
key_sredo	**kRDO**	**%g**	KEY_SREDO, 0620, Sent by shifted redo key
key_sreplace	**kRPL**	**%h**	KEY_SREPLACE, 0621, Sent by shifted replace key
key_sright	**kRIT**	**%i**	KEY_SRIGHT, 0622, Sent by shifted right-arrow key
key_srsume	**kRES**	**%j**	KEY_SRSUME, 0623, Sent by shifted resume key
key_ssave	**kSAV**	**!1**	KEY_SSAVE, 0624, Sent by shifted save key
key_ssuspend	**kSPD**	**!2**	KEY_SSUSPEND, 0625, Sent by shifted suspend key
key_stab	**khts**	**kT**	KEY_STAB, 0524, Sent by set-tab key
key_sundo	**kUND**	**!3**	KEY_SUNDO, 0626, Sent by shifted undo key
key_suspend	**kspd**	**&7**	KEY_SUSPEND, 0627, Sent by suspend key
key_undo	**kund**	**&8**	KEY_UNDO, 0630, Sent by undo key
key_up	**kcuu1**	**ku**	KEY_UP, 0403, 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_f4	**lf4**	**l4**	Labels on function key f4 if not f4
lab_f5	**lf5**	**l5**	Labels on function key f5 if not f5
lab_f6	**lf6**	**l6**	Labels on function key f6 if not f6
lab_f7	**lf7**	**l7**	Labels on function key f7 if not f7
lab_f8	**lf8**	**l8**	Labels on function key f8 if not f8
lab_f9	**lf9**	**l9**	Labels on function key f9 if not f9
lab_f10	**lf10**	**la**	Labels on function key f10 if not f10
label_off	**rmln**	**LF**	Turn off soft labels
label_on	**smln**	**LO**	Turn on soft labels
meta_off	**rmm**	**mo**	Turn off "meta mode"
meta_on	**smm**	**mm**	Turn on "meta mode" (8th bit)
newline	**nel**	**nw**	NEWLINE (behaves like **cr** followed by **lf**)
pad_char	**pad**	**pc**	Pad character (rather than null)
parm_dch	**dch**	**DC**	Delete #1 chars (G*)
parm_delete_line	**dl**	**DL**	Delete #1 lines (G*)
parm_down_cursor	**cud**	**DO**	Move cursor down #1 lines. (G*)
parm_ich	**ich**	**IC**	Insert #1 blank chars (G*)

parm_index	**indn**	**SF**	Scroll forward #1 lines. (G)
parm_insert_line	**il**	**AL**	Add #1 new blank lines (G*)
parm_left_cursor	**cub**	**LE**	Move cursor left #1 spaces (G)
parm_right_cursor	**cuf**	**RI**	Move cursor right #1 spaces. (G*)
parm_rindex	**rin**	**SR**	Scroll backward #1 lines. (G)
parm_up_cursor	**cuu**	**UP**	Move cursor up #1 lines. (G*)
pkey_key	**pfkey**	**pk**	Prog funct key #1 to type string #2
pkey_local	**pfloc**	**pl**	Prog funct key #1 to execute string #2
pkey_xmit	**pfx**	**px**	Prog funct key #1 to xmit string #2
plab_norm	**pln**	**pn**	Prog label #1 to show string #2
print_screen	**mc0**	**ps**	Print contents of the screen
prtr_non	**mc5p**	**pO**	Turn on the printer for #1 bytes
prtr_off	**mc4**	**pf**	Turn off the printer
prtr_on	**mc5**	**po**	Turn on the printer
repeat_char	**rep**	**rp**	Repeat char #1 #2 times (G*)
req_for_input	**rfi**	**RF**	Send next input char (for ptys)
reset_1string	**rs1**	**r1**	Reset terminal completely to sane modes
reset_2string	**rs2**	**r2**	Reset terminal completely to sane modes
reset_3string	**rs3**	**r3**	Reset terminal completely to sane modes
reset_file	**rf**	**rf**	Name of file containing reset string
restore_cursor	**rc**	**rc**	Restore cursor to position of last sc
row_address	**vpa**	**cv**	Vertical position absolute (G)
save_cursor	**sc**	**sc**	Save cursor position
scroll_forward	**ind**	**sf**	Scroll text up
scroll_reverse	**ri**	**sr**	Scroll text down
set_attributes	**sgr**	**sa**	Define the video attributes #1-#9 (G)
set_left_margin	**smgl**	**ML**	Set soft left margin
set_right_margin	**smgr**	**MR**	Set soft right margin
set_tab	**hts**	**st**	Set a tab stop in all rows, current column
set_window	**wind**	**wi**	Current window is lines #1-#2 cols #3-#4 (G)
tab	**ht**	**ta**	Move the cursor to the next 8 space hardware tab stop
to_status_line	**tsl**	**ts**	Go to status line, col #1 (G)
underline_char	**uc**	**uc**	Underscore one char and move past it
up_half_line	**hu**	**hu**	Half-line up (reverse 1/2 line-feed)
xoff_character	**xoffc**	**XF**	X-off character
xon_character	**xonc**	**XN**	X-on character

SAMPLE ENTRY

The following entry, which describes the Concept 100 terminal, is among the more complex entries in the **terminfo** file as of this writing.

```
concept100|c100|concept|c104|c100-4p|concept 100,
    am, db, eo, in, mir, ul, xenl, cols#80, lines#24, pb#9600, vt#8,
    bel=^G, blank=\EH, blink=\EC, clear=^L$<2*>, cnorm=\Ew, cr=^M$<9>,
    cub1=^H, cud1=^J, cuf1=\E=, cup=\Ea%p1%' '%+%c%p2%' '%+%c, cuu1=\E;,
    cvvis=\EW, dch1=\E^A$<16*>, dim=\EE, dl1=\E^B$<3*>,
    ed=\E^C$<16*>, el=\E^U$<16>, flash=\Ek$<20>\EK, ht=\t$<8>,
    il1=\E^R$<3*>, ind=^J, .ind=^J$<9>, ip=$<16*>,
    is2=\EU\Ef\E7\E5\E8\El\ENH\EK\E\0\Eo&\0\Eo\47\E,
    kbs=^h, kcub1=\E>, kcud1=\E<, kcuf1=\E=, kcuu1=\E;, kf1=\E5,
    kf2=\E6, kf3=\E7, khome=\E?, prot=\EI,
    rep=\Er%p1%c%p2%' '%+%c$<.2*>, rev=\ED,
    rmcup=\Ev\s\s\s\s$<6>\Ep\r\n, rmir=\E\0, rmkx=\Ex,
    rmso=\Ed\Ee, rmul=\Eg, rmul=\Eg, sgr0=\EN\0,
    smcup=\EU\Ev\s\s8p\Ep\r, smir=\E^P, smkx=\EX, smso=\EE\ED,
    smul=\EG,
```

Entries may continue onto multiple lines by placing white space at the beginning of each line except the first. Lines beginning with # are taken as comment lines. Capabilities in **terminfo** are of three types: boolean capabilities which indicate that the terminal has some particular feature, numeric capabilities giving the size of the terminal or particular features, and string capabilities, which give a sequence which can be used to perform particular terminal operations.

Types of Capabilities

All capabilities have names. For instance, the fact that the Concept has *automatic margins* (that is, an automatic RETURN and LINEFEED when the end of a line is reached) is indicated by the capability **am**. Hence the description of the Concept includes **am**. Numeric capabilities are followed by the character # and then the value. Thus **cols**, which indicates the number of columns the terminal has, gives the value **80** for the Concept. The value may be specified in decimal, octal or hexadecimal using normal C conventions.

Finally, string-valued capabilities, such as **el** (clear to end of line sequence) are given by the two- to five-character capname, an '=', and then a string ending at the next following comma. A delay in milliseconds may appear anywhere in such a capability, enclosed in **$<..>** brackets, as in '**el=\EK$<3>**', and padding characters are supplied by **tputs**( ) (see **curses**(3V)) to provide this delay. The delay can be either a number, for example, **20**, or a number followed by an * (for example, **3***), a / (for example, **5/**), or both (for example, **10*/**). A * indicates 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 character, the factor is still the number of lines affected. This is always one unless the terminal 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. Otherwise, if the terminal has **xon** defined, the padding information is advisory and will only be used for cost estimates or when the terminal is in raw mode. Mandatory padding will be transmitted regardless of the setting of **xon**.

A number of escape sequences are provided in the string-valued capabilities for easy encoding of characters there:

\E, \e	map to ESC
^X	maps to CTRL-*X* for any appropriate character *X*
\n	maps to NEWLINE
\l	maps to LINEFEED
\r	maps to RETURN
\t	maps to TAB
\b	maps to BACKSPACE
\f	maps to FORMFEED
\s	maps to SPACE
\0	maps to NUL

(**\0** will actually produce **\200**, which does not terminate a string but behaves as a null character on most terminals.) Finally, characters may be given as three octal digits after a backslash (for example, **\123**), and the characters ^ (caret), \ (backslash), : (colon), and , (comma) may be given as **\^**, ****, **\:**, and **\,** respectively.

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: 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 terminal description is by imitating the description of a similar terminal in **terminfo** and to build up a description gradually, using partial descriptions with some *curses*-based application to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the **terminfo** file to describe it or bugs in the application. To test a new terminal description, set the environment variable **TERMINFO** to a 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 terminal manufacturer did not document it) a severe test is to insert 16 lines into the middle of a full screen at 9600 baud. If the display is corrupted, more padding is usually needed. A similar test can be used for insert-character.

Basic Capabilities

The number of columns on each line for the terminal is given by the **cols** numeric capability. If the terminal has a screen, then the number of lines on the screen is given by the **lines** capability. If the terminal 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 terminal is a printing terminal, with no soft copy unit, give it both **hc** and **os**. (**os** applies to storage scope terminals, such as Tektronix 4010 series, as well as hard-copy and APL terminals.) If there is a code to move the cursor to the left edge of the current row, give this as **cr**. (Normally this will be RETURN, CTRL-M.) If there is a code to produce an audible signal (bell, beep, etc) give this as **bel**. If the terminal uses the xon-xoff flow-control protocol, like most terminals, specify **xon**.

If there is a code to move the cursor one position to the left (such as backspace) that capability should be given as **cub1**. Similarly, codes to move to the right, up, and down should be given as **cuf1**, **cuu1**, and **cud1**. These local cursor motions should 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 given, and should never attempt to go up locally off the top. In order to scroll text up, a program will go to the bottom left corner of the screen and send 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** which have the same semantics as **ind** and **ri** except that they take one parameter, and scroll that many lines. 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. The only local motion which is defined from the left edge is if **bw** is given, then a **cub1** from the left edge 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 terminal has switch selectable automatic margins, the **terminfo** file usually assumes that this is on; that is, **am**. If the terminal has a command which 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 terminal 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 model 33 teletype is described as

```
33|tty33|tty|model 33 teletype,
        bel=^G, cols#72, cr=^M, cud1=^J, hc, ind=^J, os,
```

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,
```

Parameterized Strings

Cursor addressing and other strings requiring parameters in the terminal are described by a parameterized string capability, with **printf**(3V)-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 it in the manner of a Reverse Polish Notation (postfix) calculator. 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. Binary operations are in postfix form with the operands in the usual order. That is, to get x–5 one would use '**%gx%{5}%–**'.

The **%** encodings have the following meanings:

%%	outputs **%**
%[[:]*flags***][***width***[.***precision***]][doxXs]**	as in **printf**(3V), flags are [–+#] and SPACE
%c	print **pop()** gives **%c**
%p[1-9]	push i^{th} parm
%P[a-z]	set variable [a-z] to **pop()**
%g[a-z]	get 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() op pop())**
%& %| %^	bit operations: **push(pop() op pop())**
%= %> %<	logical operations: **push(pop() op pop())**
%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 in 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: 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'**. Terminals which 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 TAB characters 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.

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: 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 terminal 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 terminal does not have **cup**, such as the Tektronix 4025.

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.)

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.

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 SPACE characters) 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 blanks) we have seen no terminals whose insert mode cannot be described with the single attribute.

terminfo can describe both terminals which 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 which 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 repeat the effects of **ich1** *n* times.

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 character 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.

Highlighting, Underlining, and Visible Bells

If your terminal has one or more kinds of display attributes, these can be represented in a number of different ways. You should choose one display form as *standout mode* (see **curses**(3V)), representing a good, high contrast, easy-on-the-eyes, format for highlighting error messages and other attention getters. (If you have a choice, reverse-video plus half-bright is good, or reverse-video alone; however, different users have

different preferences on different terminals.) 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 blanks on the screen, as the TVI 912 and Teleray 1061 do, then **xmc** should be given to tell how many blanks are left.

Codes to begin underlining and end underlining can be given as **smul** and **rmul** respectively. If the terminal has a code to underline the current character and move the cursor one position to the right, such as the Micro-Term MIME, this can be given as **uc**.

Other capabilities to enter various highlighting modes include **blink** (blinking), **bold** (bold or extra-bright), **dim** (dim or half-bright), **invis** (blanking or invisible text), **prot** (protected), **rev** (reverse-video), **sgr0** (turn off all attribute modes), **smacs** (enter alternate-character-set mode), and **rmacs** (exit alternate-character-set mode). Turning on any of these modes singly may or may not turn off other modes. If a command is necessary before alternate character set mode is entered, give the sequence in **enacs** (enable alternate-character-set mode).

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 be supported by **sgr**, only those for which corresponding separate attribute commands exist. (See the example at the end of this section.)

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 the terminal 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 like the Concept with more than one page of memory. If the terminal has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the terminal 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**.

If your terminal generates underlined characters by using the underline character (with no special codes needed) even though it does not otherwise overstrike characters, then you should give the capability **ul**. For terminals where a character overstriking another leaves both characters on the screen, give the capability **os**. If overstrikes are erasable with a blank, then this should be indicated by giving **eo**.

Example of highlighting: assume that the terminal under 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: 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, since 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 does not have *protect* mode, either, but that cannot be simulated in any way, so **p8** is ignored. The *altcharset* mode is different in that it is either **^O** or **^N** depending on whether it is off or on. If all modes were to be turned on, the sequence would be '**\E[0;3;4;5;7;8m^N**'.

Now look at when different sequences are output. For example, '**;3**' is output when either '**p2**' or '**p6**' is true, that is, if either *underline* or *bold* modes are turned on. Writing out the above sequences, along with their dependencies, gives the following:

sequence	**when to output**	**terminfo translation**
\E[0	always	\E[0
;3	if p2 or p6	%?%p2%p6%|%t;3%;
;4	if p1 or p3 or p6	%?%p1%p3%|%p6%|%t;4%;
;5	if p4	%?%p4%t;5%;
;7	if p1 or p5	%?%p1%p5%|%t;7%;
;8	if p7	%?%p7%t;8%;
m	always	m
^N or ^O	if p9 ^N, else ^O	%?%p9%t^N%e^O%;

Putting this all together into the **sgr** sequence gives:

sgr=\E[0%?%p2%p6%\|%t;3%;%?%p1%p3%\|%p6%\|%t;4%;%?%p5%t;5%;%?%p1%p5%
\|%t;7%;%?%p7%t;8%;m%?%p9%t^N%e^O%;,

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note: it is not possible to handle terminals 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, give these codes as **smkx** and **rmkx**. Otherwise the keypad is assumed to always transmit.

The codes 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 codes they send can be given 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 tab stops), **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 given as **pfkey**, **pfloc**, and **pfx**. A string to program their soft-screen labels can be given as **pln**. Each of these strings takes two parameters: the function key number to program (from 0 to 10) and the string to program it with. Function key numbers out of this range may

program undefined keys in a terminal-dependent manner. The difference between the capabilities is that **pfkey** causes pressing the given key to be the same as the user typing the given string; **pfloc** executes the string by the terminal in local mode; and **pfx** transmits the string to the computer. The capabilities **nlab**, **lw** and **lh** define how many soft labels there are 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.

Tabs and Initialization

If the terminal has hardware tab stops, the command to advance to the next tab stop can be given as **ht** (usually CTRL-I). A "backtab" command which moves leftward to the next tab stop can be given as **cbt**. By convention, if the teletype modes indicate that TAB characters are being expanded by the computer rather than being sent to the terminal, programs should not use **ht** or **cbt** even if they are present, since the user may not have the tab stops properly set. If the terminal has hardware tab stops which are initially set every *n* spaces when the terminal is powered up, the numeric parameter **it** is given, showing the number of spaces the tab stops are set to. This is normally used by '**tput init**' (see **tput**(1V)) to determine whether to set the mode for hardware TAB expansion and whether to set the tab stops. If the terminal 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 terminal; **iprog**, the path name of a program to be run to initialize the terminal; and **if**, the name of a file containing long initialization strings. These strings are expected to set the terminal into modes consistent with the rest of the **terminfo** description. They must be sent to the terminal 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 tab stops using **tbc** and **hts**; print the file **if**; and finally output **is3**. This is usually done using the **init** option of **tput**(1V).

Most initialization is done with **is2**. Special terminal 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 harder 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 since the terminal is usually already in 80-column mode.

If a more complex sequence is needed to set the tab stops than can be described by using **tbc** and **hts**, the sequence can be placed in **is2** or **if**.

If there are commands to set and clear margins, they can be given as **mgc** (clear all margins), **smgl** (set left margin), and **smgr** (set right margin).

Delays

Certain capabilities control padding in the terminal 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**.

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**.

Line Graphics

If the terminal has a line drawing alternate character set, the mapping of glyph to character would be given in **acsc**. The definition of this string is based on the alternate character set used in the DEC VT100 terminal, extended slightly with some characters from the AT&T 4410v1 terminal.

glyph name	VT100+ character
arrow pointing right	+
arrow pointing left	,
arrow pointing down	.
solid square block	0
lantern symbol	I
arrow pointing up	–
diamond	‘
checker board (stipple)	a
degree symbol	f
plus/minus	g
board of squares	h
lower right corner	j
upper right corner	k
upper left corner	l
lower left corner	m
plus	n
scan line 1	o
horizontal line	q
scan line 9	s
left tee (├)	t
right tee (┤)	u
bottom tee (┴)	v
top tee (┬)	w
vertical line	x
bullet	~

The best way to describe a new terminal's line graphics set is to add a third column to the above table with the characters for the new terminal that produce the appropriate glyph when the terminal is in the alternate character set mode. For example,

glyph name	VT100+ char	new tty char
upper left corner	l	R
lower left corner	m	F
upper right corner	k	T
lower right corner	j	G
horizontal line	q	,
vertical line	x	.

Now write down the characters left to right, as in '**acsc=lRmFkTjGq\,x.**'.

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 CTRL-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. On some UNIX systems, when the environment variable CC is set to a single-character value, all occurrences of the prototype character are replaced with that character.

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 UNIX 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 terminal 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 (CTRL-S and CTRL-Q, respectively), 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.

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 which can not display tilde (~) characters, such as certain Hazeltine terminals, should indicate **hz**.

Terminals which 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 TAB characters). 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 instead necessary to use delete and insert line.

Those Beehive Superbee terminals which do not transmit the escape or CTRL-C characters, should specify **xsb**, indicating that the f1 key is to be used for escape and the f2 key for CTRL-C.

Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **use** can be given with the name of the similar terminal. The capabilities given before **use** override those in the terminal type invoked by **use**. A capability can be canceled by placing *xx*@ to the left of the capability definition, where *xx* is the capability. For example, the entry

```
att4424-2|Teletype 4424 in display function group ii,
        rev@, sgr@, smul@, use=att4424,
```

defines an AT&T 4424 terminal that does not have the **rev**, **sgr**, and **smul** capabilities, and hence cannot do highlighting. This is useful for different modes for a terminal, or for different user preferences. More than one **use** capability may be given.

FILES

/usr/share/lib/terminfo/?/*	compiled terminal description database
/usr/share/lib/tabset/*	tab stop settings for some terminals, in a format appropriate to be output to the terminal (escape sequences that set margins and tab stops)

SEE ALSO

tput(1V), **curses**(3V), **printf**(3V), **term**(5V), **captoinfo**(8V), **infocmp**(8V), **tic**(8V)

WARNING

As described in the **Tabs and Initialization** section above, a terminal's initialization strings, **is1**, **is2**, and **is3**, if defined, must be output before a **curses**(3V) program is run. An available mechanism for outputting such strings is **tput init** (see **tput**(1V)).

Tampering with entries in **/usr/share/lib/terminfo/?/*** (for example, changing or removing an entry) can affect programs that expect the entry to be present and correct. In particular, removing the description for the "dumb" terminal will cause unexpected problems.

NAME

toc – table of contents of optional clusters in Application SunOS and Developer's Toolkit

SYNOPSIS

/usr/lib/load/toc

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

The **toc** file contains information specifying the organization of the optional clusters in Application SunOS and Developer's Toolkit on the Sun386i distribution media. For each cluster, a single line should be present with the following information:

cluster name
set containing the cluster (Application SunOS or Developer's Toolkit)
size of the cluster (in kilobytes)
diskette volume of the cluster in the set (for loading from 3.5" diskette)
tape and file number of the cluster (for loading from 1/4" tape)

Items are separated by a ':'.

Cluster names can contain any printable character other than a ':', space, tab, or newline character. The set containing the cluster is specified by an 'A' for Application SunOS or 'D' for Developer's Toolkit. The diskette volume is the number of the diskette within the diskette set on which the cluster begins. The tape and file number specifies the tape and file position of the cluster on the tape.

EXAMPLE

The following is an example to the **toc** file.

```
accounting:A:55:14:1@12
advanced_admin:A:628:14:1@4
audit:A:144:14:1@8
comm:A:312:13:1@9
disk_quotas:A:56:14:1@11
doc_prep:A:790:13:1@10
extended_commands:A:276:13:1@5
games:A:2351:19:1@17
mail_plus:A:135:14:1@7
man_pages:A:5586:16:1@14
name_server:A:339:14:1@13
networking_plus:A:610:13:1@6
old:A:131:14:1@16
plot:A:227:14:1@14
spellcheck:A:455:13:1@2
sysV_commands:A:2505:14:1@3
base_devel:D:5389:1:2@2
plot_devel:D:247:5:2@3
sccs:D:328:5:2@4
sunview_devel:D:1768:5:2@5
sysV_devel:D:4287:3:2@6
proflibs:D:4755:4:2@7
config:D:3065:6:2@8
```

The first line specifies that the **accounting** cluster is part of Application SunOS and requires 55 kilobytes of disk storage. In the diskette distribution, it begins on diskette 14 of Application SunOS optional clusters. In the tape distribution, it can be found on file 12 of tape 1. The last line specifies that the *config* cluster is part of Developer's Toolkit and requires 3065 kilobytes of disk storage. In the diskette distribution, it begin on diskette 6 of Developer's Toolkit. In the tape distribution, it can be found on file 8 of tape 2.

FILES

/usr/lib/load/toc

SEE ALSO

cluster(1) **load**(1) **unload**(1)

NAME

translate – input and output files for system message translation

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

These files are used by **syslogd**(8) to translate systems messages. The input file is used to map system messages (in **printf**(3V) format strings) to numbers. This number is then used to locate a new string in the output file.

An initial part of each line in the input file may specify that the message should be suppressed. Recognized suppression specifications are:

(NONE) Suppress the message always.
(n) Allow only one message every n seconds. ((10) for example).
() Do not suppress the message. This can be used in a message that begins with a '('.

Note that the message suppression specification is optional. If not present, the message is not suppressed.

Each line in the output file translates the numbers from the input file into the desired error messages, and also specifies the format to be used to output each message. The order of parameters passed from the input message can be changed, by replacing the % of a format phrase with a *%num*$ where *num* is a digit string. For example, if *num* is 2, the second parameter on the input file line will be used. The value of *num* can be from 1 to the number of parameters in the input message.

If a string is translated to a number that is not found in the output file, the message is suppressed.

EXAMPLES

An example input file:

```
$quote "
1       "(NONE)(1) logopen test code: %s\n"
2       "(10)(2) logopen test code: %s\n"
3       "()(3) logopen test code: %s\n"
4       "()(4) logopen test code: %s\n"
5       "(10)(5) logopen testcode: %s * 100\n"
6       "(10)(6) logopen testcode: %s * 100\n"
7       "(10)(7) logopen testcode: %s * 100\n"
8       "(10)%s: %s\n"
9       "(10)\n%s: write failed, file system is full\n"
10      "(10)NFS server %s not responding still trying\n"
11      "(10)NFS %s failed for server %s: %s\n"
12      "(10)NFS server %s ok\n"
13      "(NONE)\n%s: write failed, file system is full\n"
14      "(10)NFS server %s not responding still trying\n"
15      "(100)NFS %s failed for server %s: %s\n"
```

An example output file:

```
$quote "
1       "TRANSLATION:(1) logopen test code: %s\n"
2       "TRANSLATION: (2) logopen test code: %s IS REALLY\n"
3       "TRANSLATION: (3) logopen test code: %s\n"
4       "TRANSLATION: (4) logopen test code: %s\n"
5       "TRANSLATION: (5) logopen testcode: %s * 100\n"
6       "TRANSLATION: (6) logopen testcode: %s * 100\n"
7       "TRANSLATION: (7) logopen testcode: %s * 100\n"
8       "TRANSLATION: %s: %s\n"
9       "TRANSLATION: \n%s: write failed, file system is full\n"
10      "TRANSLATION: NFS server %s not responding still trying\n"
11      "TRANSLATION: NFS %s failed for server %s: %s\n"
12      "TRANSLATION: NFS server %s ok\n"
13      "Out of disk on file system %s\n"
14      "Network file server %s not ok. Check your cable\n"
15      "Network file server %2$s down (%1$s, %3$s)\n"
```

SEE ALSO

syslogd(8)

NAME

ttytab, ttys – terminal initialization data

DESCRIPTION

The **/etc/ttytab** file contains information that is used by various routines to initialize and control the use of terminal special files. This information is read with the **getttyent**(3) library routines. There is one line in **/etc/ttytab** file per special file.

The **/etc/ttys** file should not be edited; it is derived from **/etc/ttytab** by **init(8)** at boot time, and is only included for backward compatibility with programs that may still require it.

Fields are separated by TAB and/or SPACE characters. Some fields may contain more than one word and should be enclosed in double quotes. Blank lines and comments can appear anywhere in the file; comments are delimited by '#' and NEWLINE. Unspecified fields default to NULL. The first field is the terminal's entry in the device directory, **/dev**. The second field of the file is the command to execute for the line, typically **getty**(8), which performs such tasks as baud-rate recognition, reading the login name, and calling **login**(1). It can be, however, any desired command, for example the start up for a window system terminal emulator or some other daemon process, and can contain multiple words if quoted. The third field is the type of terminal normally connected to that tty line, as found in the **termcap**(5) data base file. The remaining fields set flags in the **ty_status** entry (see **getttyent**(3)) or specify a window system process that **init**(8) will maintain for the terminal line.

As flag values, the strings **on** and **off** specify whether **init** should execute the command given in the second field, while **secure** in addition to **on** allows "root" to login on this line. If the console is not marked "secure," the system prompts for the root password before coming up in single-user mode. **local** in addition to **on** indicates that the line is a "local" line; the modem control signals for this line, such as Carrier Detect, will be ignored. These flag fields should not be quoted. The string **window=** is followed by a quoted command string which **init** will execute before starting **getty**.

The flag **local** applies to terminals, and enables the software carrier mode in the kernel; the kernel ignores the state of carrier detect when opening the serial port. Alternately, if this field is set to any value other than **local**, this flag disables the software carrier mode in the kernel, so the state of the carrier detect is not ignored. This usually applies to modems. See **termio**(4).

If the line ends in a comment, the comment is included in the **ty_comment** field of the **ttyent** structure.

After changing the **/etc/ttytab** file, you must notify **init**(8) before those changes will take effect. To do this, use:

kill –1 1

EXAMPLES

Below is a sample /etc/ttytab file:

```
console "/usr/etc/getty std.1200"  vt100      on secure
ttyd0   "/usr/etc/getty d1200"     dialup     on          # 555-1234
ttyh0   "/usr/etc/getty std.9600"  hp2621-nl  on          # 254MC
ttyh1   "/usr/etc/getty std.9600"  plugboard  on          # John's office
ttyp0   none                       network
ttyp1   none                       network    off
ttyv0   "/usr/new/xterm –L :0"     vs100      on window="/usr/new/Xvs100 0"
console "/usr/etc/getty –n –s   std.9600" sun      on    secure
console "/usr/etc/getty –n –s –l std.9600" sun      on    secure
```

The first line permits "root" login on the console at 1200 baud, and indicates that the console is physically secure for single-user operation. The second line allows dialup at 1200 baud without "root" login, and the third and fourth lines allow login at 9600 baud with terminal types of **hp2621-nl** and **plugboard**, respectively. The fifth and sixth lines are examples of network pseudo-ttys, **ttyp0** and **ttyp1** for which **getty** should not be enabled. The seventh line shows a terminal emulator and window-system startup entry. The last two lines instruct **getty**, using the **–n** argument, to run the **logintool**(8) graphic login interface, and the **–s** argument instructing **logintool** to start **screenblank**(1) with a plain black screen. The **–l** (lower case L) argument instructs **logintool** to start **lockscreen**(1). **lockscreen** starts after 30 minutes; there is no way to change this interval.

FILES

/dev
/etc/ttys
/etc/ttytab

SEE ALSO

login(1), **ioctl**(2), **getttyent**(3), **termio**(4), **gettytab**(5), **termcap**(5), **getty**(8), **init**(8), **logintool**(8), **ttysoftcar**(8)

NAME

types – primitive system data types

SYNOPSIS

#include <sys/types.h>

DESCRIPTION

The data types defined in the include file are used in the system code; some data of these types are accessible to user code:

```
/*
 * Copyright (c) 1982, 1986 Regents of the University of California.
 * All rights reserved.  The Berkeley software License Agreement
 * specifies the terms and conditions for redistribution.
 */

#ifndef _TYPES_
#define _TYPES_

/*
 * Basic system types.
 */

#include <sys/sysmacros.h>

typedef unsigned char   u_char;
typedef unsigned short  u_short;
typedef unsigned int    u_int;
typedef unsigned long   u_long;
typedef unsigned short  ushort;/* System V compatibility */
typedef unsigned int    uint;/* System V compatibility */

#ifdef vax
typedef struct      _physadr { int r[1]; } *physadr;
typedef struct      label_t{
        int         val[14];
} label_t;
#endif
#ifdef mc68000
typedef struct      _physadr { short r[1]; } *physadr;
typedef struct      label_t{
        int         val[13];
} label_t;
#endif
#ifdef sparc
typedef struct _physadr { int r[1]; } *physadr;
typedef struct label_t {
        int    val[2];
} label_t;
#endif
#ifdef i386
typedef struct      _physadr { short r[1]; } *physadr;
typedef struct      label_t {
        int         val[8];
} label_t;
```

```
#endif
typedef struct  _quad { long val[2]; } quad;
typedef long    daddr_t;
typedef char *  caddr_t;
typedef u_long  ino_t;
typedef long    swblk_t;
typedef int     size_t;
typedef long    time_t;
typedef short   dev_t;
typedef long    off_t;
typedef u_short uid_t;
typedef u_short gid_t;
typedef long    key_t;

#define NBBY    8       /* number of bits in a byte */
/*
 * Select uses bit masks of file descriptors in longs.
 * These macros manipulate such bit fields (the filesystem macros use chars).
 * FD_SETSIZE may be defined by the user, but the default here
 * should be >= NOFILE (param.h).
 */
#ifndef FD_SETSIZE
#define FD_SETSIZE  256
#endif

typedef long    fd_mask;
#define NFDBITS     (sizeof(fd_mask) * NBBY)/* bits per mask */
#ifndef howmany
#ifdef sun386
#define howmany(x, y)   ((((u_int)(x))+(((u_int)(y))-1))/((u_int)(y)))
#else
#define howmany(x, y)   (((x)+((y)-1))/(y))
#endif
#endif

typedef struct fd_set {
        fd_mask fds_bits[howmany(FD_SETSIZE, NFDBITS)];
} fd_set;

typedef char *  addr_t;

#define FD_SET(n, p)    ((p)->fds_bits[(n)/NFDBITS] |= (1 << ((n) % NFDBITS)))
#define FD_CLR(n, p)    ((p)->fds_bits[(n)/NFDBITS] &= ~(1 << ((n) % NFDBITS)))
#define FD_ISSET(n, p) ((p)->fds_bits[(n)/NFDBITS] & (1 << ((n) % NFDBITS)))
#define FD_ZERO(p)      bzero((char *)(p), sizeof(*(p)))

#ifdef sparc
/*
 * routines that call setjmp have strange control flow graphs,
 * since a call to a routine that calls resume/longjmp will eventually
 * return at the setjmp site, not the original call site.  This
 * utterly wrecks control flow analysis.
 */
```

```
extern int setjmp();
#pragma unknown_control_flow(setjmp)
#endif sparc

#endif  _TYPES_
```

The form *daddr_t* is used for disk addresses, see **fs**(5). Times are encoded in seconds since 00:00:00 GMT, January 1, 1970. The major and minor parts of a device code specify kind and unit number of a device and are installation-dependent. Offsets are measured in bytes from the beginning of a file. The *label_t* variables are used to save the processor state while another process is running.

SEE ALSO

adb(1), **lseek**(2V), **time**(3V), **fs**(5)

NAME

tzfile – time zone information

SYNOPSIS

#include <tzfile.h>

DESCRIPTION

The time zone information files used by **tzset** (see **ctime**(3V)) begin with bytes reserved for future use, followed by three four-byte values of type **long**, written in a "standard" byte order (the high-order byte of the value is written first). These values are, in order:

tzh_timecnt	The number of "transition times" for which data is stored in the file.
tzh_typecnt	The number of "local time types" for which data is stored in the file (must not be zero).
tzh_charcnt	The number of characters of "time zone abbreviation strings" stored in the file.

The above header is followed by *tzh_timecnt* four-byte values of type **long**, sorted in ascending order. These values are written in "standard" byte order. Each is used as a transition time (as returned by **gettimeofday**(2)) at which the rules for computing local time change. Next come *tzh_timecnt* one-byte values of type **unsigned char**; each one tells which of the different types of "local time" types described in the file is associated with the same-indexed transition time. These values serve as indices into an array of *ttinfo* structures that appears next in the file; these structures are defined as follows:

```
struct ttinfo {
        long            tt_gmtoff;
        int             tt_isdst;
        unsigned int    tt_abbrind;
};
```

Each structure is written as a four-byte value for *tt_gmtoff* of type **long**, in a standard byte order, followed by a one-byte value for *tt_isdst* and a one-byte value for *tt_abbrind*. In each structure, *tt_gmtoff* gives the number of seconds to be added to GMT, *tt_isdst* tells whether *tm_isdst* should be set by **localtime** (see **ctime**(3V)) and *tt_abbrind* serves as an index into the array of time zone abbreviation characters that follow the *ttinfo* structure(s) in the file.

localtime uses the first standard-time *ttinfo* structure in the file (or simply the first *ttinfo* structure in the absence of a standard-time structure) if either *tzh_timecnt* is zero or the time argument is less than the first transition time recorded in the file.

SEE ALSO

gettimeofday(2), **ctime**(3V)

NAME

ugid_alloc.range – range of user IDs and group IDs to allocate

SYNOPSIS

/etc/ugid_alloc.range

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

The **/etc/ugid_alloc.range** file, if it exists on the Network Information Service (NIS) master of the **passwd.byuid** map (or the **group.bygid** map for group IDs), specifies the user IDs and group IDs that can be allocated for the local NIS domain by the **uid_allocd**(8C) daemons. If the file does not exist, user IDs or group IDs may be allocated beginning at 100 and ending at 60,000; no user IDs or group IDs are allocated out of that range in any case. If the local NIS domain is not listed in this file, no user IDs or group IDs will be allocated. Otherwise, this file specifies ranges of user IDs or group IDs that may be allocated. The different NIS domains on a network can use identical copies of this file.

If a network has multiple NIS domains, each one will typically use ranges for its user IDs and group IDs that do not overlap with the other NIS domains, guaranteeing that user IDs and group IDs are unique throughout the network. Without guarantees of user ID and group ID uniqueness, network tools and services which rely on that uniqueness for security or authentication will not work as intended. Such services include NFS, except for the "Secure NFS," which has other solutions for security and authentication. Note: the required uniqueness could be guaranteed by mechanisms other than automatic allocation within manually configured ranges. For example, some sites can use a function of their employee numbers during manual user ID allocation, and coordinate group ID assignment verbally.

This file can contain blank lines. Comments begin with a '#' character and extend to the end of the current line. The first token on the line is an NIS domain name. It is separated from the second token by white space (SPACE or TAB characters). The second token is either *user* or *group*, indicating that the line specifies user ID or group ID ranges, respectively. The third token is a comma-separated list of user or group ID ranges in that domain. These ranges take two forms: a single number specifies just that ID, and two numbers separated by a dash specify all IDs starting at the first number and ending with the second.

For example, the following file would direct that the manufacturing department at a particular company use user IDs from 700 to 999 or 1200 to 1499. Accounts created by tools in the NIS domain for manufacturing would use a user ID in those ranges, and those user accounts could safely be added to one of the other NIS domains if desired (by manually transferring NIS map data between the domains). Group IDs are allocated only within the administration domain.

```
# Three departments share our site's network, and each has its
# own Ethernet and master server connected with IP routers.
# This file sets the user ID ranges assigned to each department.
# Groups are defined by the administration group only.
YP.admin.company.com              user    500-699
YP.manufacturing.company.com      user    700-999
YP.engineering.company.com        user    100-499,1000-1199
YP.manufacturing.company.com      user    1200-1499
YP.admin.company.com              group   100-60000
```

SEE ALSO

passwd(5), **group**(5), **uid_allocd**(8C)

BUGS

There is a limit of forty ranges for each domain; more ranges are silently ignored.

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.

NAME

updaters – configuration file for NIS updating

SYNOPSIS

/var/yp/updaters

DESCRIPTION

The file **/var/yp/updaters** is a makefile (see **make**(1)) 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**(5) 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)

- Network name of client wishing to make the update (a string)
- Kind of update (an integer)
- Number of bytes in key (an integer)
- Actual bytes of key
- Number of bytes in data (an integer)
- Actual bytes of data

After getting this information through standard input, the command to update the particular database should decide whether the user is allowed to make the change. If not, it should exit with the status YPERR_ACCESS. If the user is allowed to make the change, the command should make the change and exit with a status of zero. If there are any errors that may prevent the updater from making the change, it should exit with the status that matches a valid NIS error code described in **<rpcsvc/ypclnt.h>**.

FILES

/var/yp/updaters

SEE ALSO

make(1), **ypupdate**(3N), **publickey**(5), **ypupdated**(8C)

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

utmp, wtmp, lastlog – login records

SYNOPSIS

#include <utmp.h>
#include <lastlog.h>

DESCRIPTION

utmp file

The **utmp** file records information about who is currently using the system. The file is a sequence of **utmp** structure entries. That structure is defined in **<utmp.h>**, and contains the following members:

ut_line	Character array containing the name of the terminal on which the user logged in.
ut_name	Character array containing the name of the user who logged in.
ut_host	Character array containing the name of the host from which the user remotely logged in, if they logged in from another host; otherwise, a null string.
ut_time	**long** containing the time at which the user logged in, in seconds since 00:00 GMT, January 1, 1970.

Whenever a user logs in, **login**(1) fills in the entry in **/etc/utmp** for the terminal on which the user logged in. When they log out, **init**(8) clears that entry by setting **ut_name** and **ut_host** to null strings and **ut_time** to the time at which the user logged out.

Some window systems will make entries in **utmp** for terminal emulation windows running shells, so that library routines such as **getlogin** will work correctly in that window. These entries do not directly represent logged-in users; they are associated with a user who has already logged into the system on another terminal. These entries generally have a **ut_line** field that refers to a pseudo-terminal, and a **ut_host** field that is a null string. The macro **nonuser**, defined in **<utmp.h>**, takes a pointer to a **utmp** structure as an argument and, if the entry has a **ut_line** field that refers to a pseudo-terminal, and a **ut_host** field that is a null string, will return 1; otherwise, it will return 0. This can be used by programs that print information about logged-in users if they should not list entries made for logged-in users' additional windows.

wtmp file

The **wtmp** file records all logins and logouts. It also consists of a sequence of **utmp** entries.

Whenever a user logs in, **login** appends a record identical to the record it placed in **utmp** to the end of **/var/adm/wtmp.** Whenever a user logs out, **init** appends a record with **ut_line** equal to the terminal that the user was logged in on, **ut_name** and **ut_host** null, and **ut_time** equal to the time at which the user logged out.

When the system is shut down, **init** appends a record with a **ut_line** of ~, a **ut_name** of **shutdown**, a null **ut_host**, and a **ut_time** equal to the time at which the shutdown occurred. When the system is rebooted, **init** appends a record with a **ut_line** of ~, a **ut_name** of **reboot**, a null **ut_host**, and a **ut_time** equal to the time at which **init** wrote the record.

When the **date** command is used to change the system-maintained time, **date** appends a record with a **ut_line** of |, **ut_name** and **ut_host** null, and **ut_time** equal to the system time before the change, and then appends a record with a **ut_line** of {, **ut_name** and **ut_host** null, and **ut_time** equal to the system time after the change.

None of the programs that maintain **wtmp** create the file, so that if record-keeping is to be enabled, it must be created by hand as a zero-length file, and if it is removed, record-keeping is turned off. It is summarized by **ac**(8).

As **wtmp** is appended to whenever a user logs in or out, it should be truncated periodically so that it does not consume all the disk space on its file system.

lastlog file

The **lastlog** file records the most recent login-date for every user logged in. The file is a sequence of **lastlog** structure entries. That structure is defined in **<lastlog.h>**, and contains the following members:

ll_time **long** containing the time at which the user logged in, in seconds since 00:00 GMT, January 1, 1970.

ll_line Character array containing the name of the terminal on which the user logged in.

ll_host Character array containing the name of the host from which the user remotely logged in, if they logged in from another host; otherwise, a null string.

When reporting (and updating) the most recent login date, **login** performs an **lseek**(2V) to a byte-offset in **/var/adm/lastlog** corresponding to the userid. Because the count of userids may be high, whereas the number actual users may be small within a network environment, the bulk of this file may never be allocated by the file system even though an offset may appear to be quite large. Although **ls**(1V) may show it to be large, chances are that this file need not be truncated. **du**(1V) will report the correct (smaller) amount of space actually allocated to it.

SYSTEM V DESCRIPTION

For XPG2 conformance, the XPG2 private **utmp** structure is preserved for use by compliant applications that specifically use the **utmp structure.** The structure is defined in **/usr/xpg2include/utmp.h**. Note: this structure definition was removed in XPG3, and will be removed in a future SunOS release. Applications using the XPG2 **utmp** structure must do so on an application private basis.

FILES

/etc/utmp
/var/adm/wtmp
/var/adm/lastlog

SEE ALSO

login(1), **who**(1), **ac**(8), **init**(8)

NAME

uuencode – format of an encoded uuencode file

DESCRIPTION

Files output by **uuencode**(1C) consist of a header line, followed by a number of body lines, and a trailer line. **uudecode** (see **uuencode**(1C)) will ignore any lines preceding the header or following the trailer. Lines preceding a header must not, of course, look like a header.

The header line is distinguished by having the first 6 characters '**begin** '. The word **begin** is followed by a mode (in octal), and a string which names the remote file. Spaces separate the three items in the header line.

The body consists of a number of lines, each at most 62 characters long (including the trailing NEWLINE). These consist of a character count, followed by encoded characters, followed by a NEWLINE. The character count is a single printing character, and represents an integer, the number of bytes the rest of the line represents. Such integers are always in the range from 0 to 63 and can be determined by subtracting the character space (octal 40) from the character.

Groups of 3 bytes are stored in 4 characters, 6 bits per character. All are offset by a SPACE to make the characters printing. The last line may be shorter than the normal 45 bytes. If the size is not a multiple of 3, this fact can be determined by the value of the count on the last line. Extra garbage will be included to make the character count a multiple of 4. The body is terminated by a line with a count of zero. This line consists of one ASCII SPACE.

The trailer line consists of **end** on a line by itself.

SEE ALSO

mail(1), **uucp**(1C), **uuencode**(1C), **uusend**(1C)

NAME

vfont – font formats

SYNOPSIS

#include <vfont.h>

DESCRIPTION

The fonts used by the window system and printer/plotters have the following format. Each font is in a file, which contains a header, an array of character description structures, and an array of bytes containing the bit maps for the characters. The header has the following format:

```
struct header {
        short           magic;          /* Magic number VFONT_MAGIC */
        unsigned short  size;           /* Total # bytes of bitmaps */
        short           maxx;           /* Maximum horizontal glyph size */
        short           maxy;           /* Maximum vertical glyph size */
        short           xtend;          /* (unused) */
};
#define VFONT_MAGIC                     0436
```

maxx and *maxy* are intended to be the maximum horizontal and vertical size of any glyph in the font, in raster lines. (A glyph is just a printed representation of a character, in a particular size and font.) The size is the total size of the bit maps for the characters in bytes. The *xtend* field is not currently used.

After the header is an array of NUM_DISPATCH structures, one for each of the possible characters in the font. Each element of the array has the form:

```
struct dispatch {
        unsigned short  addr;           /* &(glyph) - &(start of bitmaps) */
        short           nbytes;         /* # bytes of glyphs (0 if no glyph) */
        char            up, down, left, right;  /* Widths from baseline point */
        short           width;          /* Logical width, used by troff */
};
#define NUM_DISPATCH                    256
```

The *nbytes* field is nonzero for characters which actually exist. For such characters, the *addr* field is an offset into the bit maps to where the character's bit map begins. The *up* , *down*, *left*, and *right* fields are offsets from the base point of the glyph to the edges of the rectangle which the bit map represents. (The imaginary ''base point'' is a point which is vertically on the ''base line'' of the glyph (the bottom line of a glyph which does not have a descender) and horizontally near the left edge of the glyph; often 3 or so pixels past the left edge.) The bit map contains *up+down* rows of data for the character, each of which has *left+right* columns (bits). Each row is rounded up to a number of bytes. The *width* field represents the logical width of the glyph in bits, and shows the horizontal displacement to the base point of the next glyph.

FILES

/usr/lib/vfont/*
/usr/lib/fonts/fixedwidthfonts/*

SEE ALSO

troff(1), **vfontinfo**(1), **vswap**(1)

BUGS

A machine-independent font format should be defined. The shorts in the above structures contain different bit patterns depending whether the font file is for use on a VAX or a Sun. The vswap program must be used to convert one to the other.

NAME

vgrindefs – vgrind's language definition data base

SYNOPSIS

/usr/lib/vgrindefs

DESCRIPTION

vgrindefs contains all language definitions for **vgrind**(1). The data base is very similar to **termcap**(5). Capabilities in **vgrindefs** are of two types: Boolean capabilities which indicate that the language has some particular feature and string capabilities which give a regular expression or keyword list. Entries may continue onto multiple lines by giving a \ as the last character of a line. Lines starting with # are comments.

Capabilities

The following table names and describes each capability.

Name	Type	Description
ab	str	Regular expression for the start of an alternate form comment
ae	str	Regular expression for the end of an alternate form comment
bb	str	Regular expression for the start of a block
be	str	Regular expression for the end of a lexical block
cb	str	Regular expression for the start of a comment
ce	str	Regular expression for the end of a comment
id	str	String giving characters other than letters and digits that may legally occur in identifiers (default '_')
kw	str	A list of keywords separated by spaces
lb	str	Regular expression for the start of a character constant
le	str	Regular expression for the end of a character constant
oc	bool	Present means upper and lower case are equivalent
pb	str	Regular expression for start of a procedure
pl	bool	Procedure definitions are constrained to the lexical level matched by the 'px' capability
px	str	A match for this regular expression indicates that procedure definitions may occur at the next lexical level. Useful for lisp-like languages in which procedure definitions occur as subexpressions of defuns.
sb	str	Regular expression for the start of a string
se	str	Regular expression for the end of a string
tc	str	Use the named entry as a continuation of this one
tl	bool	Present means procedures are only defined at the top lexical level

Regular Expressions

vgrindefs uses regular expressions similar to those of **ex**(1) and **lex**(1). The characters '^', '$', ':', and '\' are reserved characters and must be 'quoted' with a preceding \ if they are to be included as normal characters. The metasymbols and their meanings are:

$	The end of a line
^	The beginning of a line
\d	A delimiter (space, tab, newline, start of line)
\a	Matches any string of symbols (like '.*' in lex)
\p	Matches any identifier. In a procedure definition (the 'pb' capability) the string that matches this symbol is used as the procedure name.
()	Grouping
|	Alternation
?	Last item is optional
\e	Preceding any string means that the string will not match an input string if the input string is preceded by an escape character (\). This is typically used for languages (like C) that can include the string delimiter in a string by escaping it.

Unlike other regular expressions in the system, these match words and not characters. Hence something like '(tramp|steamer)flies?' would match 'tramp', 'steamer', 'trampflies', or 'steamerflies'. Contrary to some forms of regular expressions, **vgrindef** alternation binds very tightly. Grouping parentheses are likely to be necessary in expressions involving alternation.

Keyword List

The keyword list is just a list of keywords in the language separated by spaces. If the 'oc' boolean is specified, indicating that upper and lower case are equivalent, then all the keywords should be specified in lower case.

EXAMPLE

The following entry, which describes the C language, is typical of a language entry.

```
C|c|the C programming language:\
        :pb=^\d?*?\d?\p\d??):bb={:be=}:cb=/*:ce=*/:sb=":se=\e":\
        :lb=':le=\e':tl:\
        :kw=asm auto break case char continue default do double else enum\
        extern float for fortran goto if int long register return short\
        sizeof static struct switch typedef union unsigned while #define\
        #else #endif #if #ifdef #ifndef #include #undef # define else endif\
        if ifdef ifndef include undef:
```

Note that the first field is just the language name (and any variants of it). Thus the C language could be specified to **vgrind**(1) as 'c' or 'C'.

FILES

/usr/lib/vgrindefs file containing terminal descriptions

SEE ALSO

troff(1), **vgrind**(1)

NAME

ypaliases – NIS aliases for sendmail

SYNOPSIS

/etc/ypaliases

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

Create the Network Information Service (NIS) aliases map with this text file. The **/etc/ypaliases** file has the same format as the **/etc/aliases** file described in **aliases**(5).

The text file for the NIS aliases map is stored in the **/etc/aliases** file on the NIS master of an NIS domain. Other systems in a domain (besides the NIS master) can also have a local **/etc/aliases** file. The local file is accessed first by programs such as **sendmail**(8), and if it contains a line beginning with the character '+', the NIS map will be accessed.

The local **/etc/aliases** file can specify resources that are not available on a network-wide basis. This implies that the NIS master cannot use the local **/etc/aliases** file to specify aliases that are to be known only to the local system. Sun386i systems allow the **/etc/aliases** file on the NIS master to be used locally, creating the NIS aliases map with the **/etc/ypaliases** text file.

FILES

/etc/aliases
/etc/ypaliases

SEE ALSO

uucp(1C), **dbm**(3X), **aliases**(5), **newaliases**(8), **sendmail**(8)

System and Network Administration

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

ypfiles – NIS database and directory structure

DESCRIPTION

The Network Information Service (NIS) uses a distributed, replicated database of **dbm** files contained in the **/var/yp** directory hierarchy on each NIS server. A **dbm** database consists of two files, created by calls to the **ndbm**(3) library package. One has the filename extension **.pag** and the other has the filename extension **.dir**. For instance, the database named **hosts.byname**, is implemented by the pair of files **hosts.byname.pag** and **hosts.byname.dir**.

A **dbm** database served by the NIS service is called an NIS *map*. An NIS *domain* is a subdirectory of **/var/yp** containing a set of NIS maps. Any number of NIS domains can exist. Each may contain any number of maps.

No maps are required by the NIS lookup service itself, although they may be required for the normal operation of other parts of the system. There is no list of maps which the NIS service serves — if the map exists in a given domain, and a client asks about it, the NIS service will serve it. For a map to be accessible consistently, it must exist on all NIS servers that serve the domain. To provide data consistency between the replicated maps, an entry to run **ypxfr** periodically should be made in the super-user's **crontab** file on each server. More information on this topic is in **ypxfr**(8).

The NIS maps should contain two distinguished key-value pairs. The first is the key **YP_LAST_MODIFIED**, having as a value a ten-character ASCII order number. The order number should be the system time in seconds when the map was built. The second key is **YP_MASTER_NAME**, with the name of the NIS master server as a value. **makedbm**(8) generates both key-value pairs automatically. A map that does not contain both key-value pairs can be served by the NIS service, but the **ypserv** process will not be able to return values for "Get order number" or "Get master name" requests. See **ypserv**(8). In addition, values of these two keys are used by **ypxfr** when it transfers a map from a master NIS server to a slave. If **ypxfr** cannot figure out where to get the map, or if it is unable to determine whether the local copy is more recent than the copy at the master, you must set extra command line switches when you run it.

The NIS maps must be generated and modified only at the master server. They are copied to the slaves using **ypxfr**(8) to avoid potential byte-ordering problems among the NIS servers running on machines with different architectures, and to minimize the amount of disk space required for the dbm files. The NIS database can be initially set up for both masters and slaves by using **ypinit**(8).

After the server databases are set up, it is probable that the contents of some maps will change. In general, some ASCII source version of the database exists on the master, and it is changed with a standard text editor. The update is incorporated into the NIS map and is propagated from the master to the slaves by running **/var/yp/Makefile**. All Sun-supplied maps have entries in **/var/yp/Makefile**; if you add an NIS map, edit this file to support the new map. The makefile uses **makedbm**(8) to generate the NIS map on the master, and **yppush**(8) to propagate the changed map to the slaves. **yppush** is a client of the map **ypservers**, which lists all the NIS servers. For more information on this topic, see **yppush**(8).

FILES

/var/yp
/var/yp/Makefile

SEE ALSO

dbm(3X), **makedbm**(8), **rpcinfo**(8C), **ypinit**(8), **ypmake**(8), **yppoll**(8), **yppush**(8), **ypserv**(8), **ypxfr**(8)

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

ypgroup – NIS group file

SYNOPSIS

/etc/ypgroup

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

Create the Network Information Service (NIS) group map with this text file. This file has the same format as the **/etc/group** file described in **group**(5).

The text file for the NIS group map is stored in the **/etc/group** file on the NIS master of an NIS domain. Other systems in a domain (besides the NIS master) can also have a local **/etc/group** file. The local file is accessed first by programs such as **groups**(1), and if it contains a line beginning with the character '+', the NIS map will be accessed. The local **/etc/group** file can specify groups that are not available on a network-wide basis.

This implies that the NIS master cannot use the local **/etc/group** file to specify groups that are to be known only to the local system. Sun386i systems allow the **/etc/group** file on the NIS master to be used locally, creating the NIS group map from the **/etc/ypgroup** text file.

FILES

/etc/group
/etc/ypgroup

SEE ALSO

passwd(1), **su**(1V), **getgroups**(2V), **crypt**(3), **initgroups**(3), **group**(5), **group.adjunct**(5), **passwd**(5), **grpck**(8V)

System and Network Administration,
Sun386i SNAP Administration,
Sun386i Advanced Administration

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

yppasswd – NIS password file

SYNOPSIS

/etc/yppasswd

DESCRIPTION

Create the Network Information Service (NIS) password map with this text file. The format for **/etc/yppasswd** is the same as for the **/etc/passwd** file described in **passwd**(5).

The text file for the NIS password map is stored in the **/etc/passwd** file on the NIS master of an NIS domain. Other systems in a domain can also have a local **/etc/passwd** file. The local file is accessed first by programs such as **passwd**(1), and if it contains a line beginning with the character '+', the NIS map will be accessed.

The local **/etc/passwd** file can specify users that are not available on a network-wide basis. This implies that the NIS master cannot use the local **/etc/passwd** file to specify users that are to be known only to the local system. Sun386i systems allow the **/etc/passwd** file on the NIS master to be used locally, creating the NIS password map from the **/etc/yppasswd** text file.

FILES

/etc/passwd
/etc/yppasswd

SEE ALSO

login(1), **mail**(1), **passwd**(1), **crypt**(3), **getpwent**(3V), **group**(5), **passwd**(5), **passwd.adjunct**(5), **adduser**(8), **sendmail**(8), **vipw**(8)

System and Network Administration,
Sun386i SNAP Administration,
Sun386i Advanced Administration

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

ypprintcap – NIS printer capability database

SYNOPSIS

/etc/ypprintcap

AVAILABILITY

Available only on Sun 386i systems running a SunOS 4.0.*x* release or earlier. Not a SunOS 4.1 release feature.

DESCRIPTION

Create the Network Information Service (NIS) printcap map with this text file to centralize and simplify printer administration. The **/etc/ypprintcap** file has the same format as the **/etc/printcap** file described in **printcap**(5).

The text file for the NIS printcap map is stored in the **/etc/printcap** file on the NIS master of an NIS domain. Other systems in a domain (besides the NIS master) can also have a local **/etc/printcap** file. The local file is accessed first by programs such as **lpr**(1), and if it contains a line beginning with the character '+', the NIS map will be accessed.

The local **/etc/printcap** file can specify printers that are not available on a network-wide basis. This implies that the NIS master cannot use the local **/etc/printcap** file to specify printers that are to be known only to the local system. Sun386i systems allow the **/etc/printcap** file on the NIS master to be used locally, using the **/etc/ypprintcap** file to create the NIS printcap map.

FILES

/etc/printcap
/etc/ypprintcap

SEE ALSO

lpq(1), **lpr**(1), **lprm**(1), **snap**(1), **stty**(1V), **plot**(3X), **ttcompat**(4M), **printcap**(5), **termcap**(5), **lpc**(8), **lpd**(8), **pac**(8)

System and Network Administration,
Sun386i SNAP Administration,
Sun386i Advanced Administration

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.