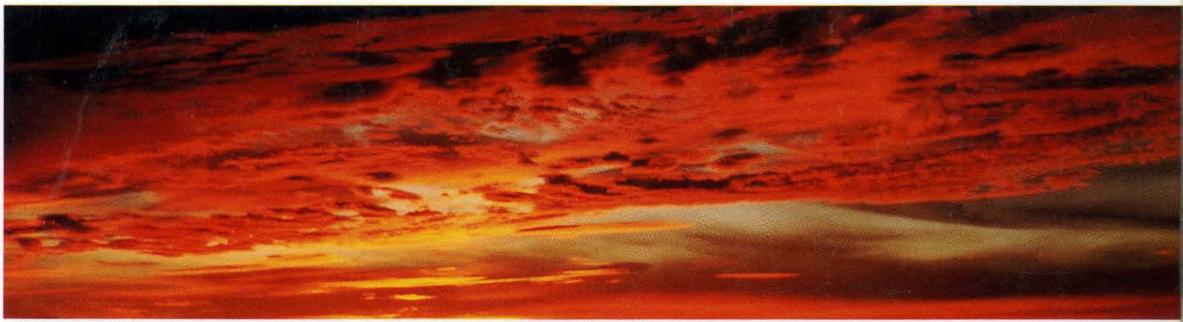
R E F E R E N C E



SYSTEM FILES AND DEVICES REFERENCE

UNIX® SVR4.2

UNIX PRESS

REFERENCE MANUAL	DESCRIPTION	SECTIONS
Command Reference (Commands a - l)	General-Purpose User Commands	**1**
	Basic Networking Commands	**1C**
	Form and Menu Language Interpreter	**1F**
	System Maintenance Commands	**1M**
Command Reference (Commands m - z)	(same as above)	
Operating System API Reference	System Calls	**2**
	BSD System Compatibility Library	**3**
	Standard C Library	**3C**
	ETI-curses Library	**3curses**
	Executable and Linking Format Library	**3E**
	General-Purpose Library	**3G**
	Identification and Authentication Library	**3I**
	Math Library	**3M**
	Networking Library	**3N**
	Standard I/O Library	**3S**
	Multibyte/Wide Character Conversion Library	**3W**
	Specialized Libraries	**3X**
Windowing System API Reference	Desktop Metaphor	**3Dt**
	Drag and Drop	**3DnD**
	MoOLIT	**3Olit**
	ETI-curses Library	**3curses**
System Files and Devices Reference	System File Formats	**4**
	Miscellaneous Facilities	**5**
	Special Files (Devices)	**7**
Device Driver Reference	DDI/DKI Driver Data Definitions	**D1**
	DDI/DKI Driver Entry Point Routines	**D2**
	DDI/DKI Kernel Utility Routines	**D3**
	Portable Device Interface (PDI) Routines	**D3G**
	SCSI Device Interface (SDI) Routines	**D3I**
	DDI/DKI Kernel Data Structures	**D4**
	SCSI Device Interface (SDI) Data Structures	**D4I**
	DDI/DKI Kernel Defines	**D5**

SYSTEM FILES AND DEVICES REFERENCE

UNIX SVR4.2

Edited by Lynda Feng

UNIX
Press

Published by Prentice-Hall, Inc.
A Simon & Schuster Company
Englewood Cliffs, New Jersey 07632

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-017682-6

UNIX
PRESS
A Prentice Hall Title

Table of Contents

Introduction

File Formats

Miscellaneous Facilities

Introduction

Computers keep track of thousands and thousands of details. To save the labor of continuously respecifying these details, information that is used repeatedly is stored in files, which the operating system references when needed. For example, when you turn the power on, the operating system reads a file that specifies which disks to mount; when you log in, it validates your password and sets up your environment; when you copy files from a remote system, it maps the software names to the network addresses.

The System Files and Devices Reference describes the system and device files in the UNIX System, including both special files and regular files. Special files pertain to a particular hardware device; regular files are hardware-independent. The book also includes a set of miscellaneous manual pages. Not all of the files and devices described in this manual are available on every UNIX system. Some of the features require additional utilities that may not exist on your system.

The System Files and Devices Reference is part of a comprehensive UNIX system reference set, which describes commands, system calls, libraries, and files. This book includes all manual pages in sections 4, 5, and 7. References to manual pages in other sections are found in other books in the reference set. The inner front cover of this book lists the various section numbers and the books in which they are found.

Manual Page Format

All manual page entries use a common format, not all of whose parts always appear:

- The **NAME** section gives the name(s) of the entry and briefly states its purpose.
- The **SYNOPSIS** section summarizes the use of the command, program or function, or names the relevant special file.
- The **DESCRIPTION** section describes the utility.
- The **EXAMPLE** section gives example(s) of usage, where appropriate.
- The **FILES** section gives the file names that are built into the program.
- The **SEE ALSO** section gives pointers to related information. Reference to manual pages with section numbers other than those in this book can be found in other reference manuals, as listed above.

- The **DIAGNOSTICS** section discusses the diagnostic indications that may be produced. Messages that are intended to be self-explanatory are not listed.
- The **NOTES** section gives generally helpful hints about the use of the utility.

Request for Comment

A Request for Comment (RFC) is a document that describes some aspect of networking technology. The RFCs cited in the SEE ALSO section of these manual pages are available in hardcopy from:

> Jon Postel
> RFC Editor
> USC Information Sciences Institute
> 4676 Admiralty Way
> Marina del Rey, CA 90292-6695

Online versions of the RFCs are available by FTP from **`nic.ddn.mil`**. To connect to this host, type:

```
ftp -n nic.ddn.mil
```

Log in with the user name **`anonymous`** and the password **`guest`**. To retrieve the RFC, type `get rfc:rfc`*num*`.txt`, where *num* is replaced by the number of the RFC. For example, to retrieve RFC 1171, type

```
get rfc:rfc1171.txt
```

At the end of the `ftp` session, type `quit` to exit.

NAME

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

SYNOPSIS

`#include <elf.h>`

DESCRIPTION

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

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

Linking View	Execution View
ELF header	ELF header
Program header table *optional*	Program header table
Section 1	Segment 1
. . .	
Section *n*	Segment 2
. . .	
. . .	. . .
Section header table	Section header table *optional*

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

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

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

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

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

SEE ALSO

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

NAME

acct – per-process accounting file format

SYNOPSIS

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

DESCRIPTION

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

```
typedef  ushort comp_t; /* "floating point" */
                        /* 13-bit fraction, 3-bit exponent */
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;     /* acctng user time in clock ticks */
   comp_t ac_stime;     /* acctng system time in clock ticks */
   comp_t ac_etime;     /* acctng elapsed time in clock ticks */
   comp_t ac_mem;       /* memory usage in clicks */
   comp_t ac_io;        /* chars trnsfrd by read/write */
   comp_t ac_rw;        /* number of block reads/writes */
   char   ac_comm[8];   /* command name */
};

extern  struct  acct  acctbuf;
extern  struct  vnode  *acctp;  /* vnode of accounting file */

#define AFORK   01      /* has executed fork, but no exec */
#define    ASU      02     /* used super-user privileges */
#define    ACCTF    0300   /* record type: 00 = acct */
#define    AEXPND   040    /* Expanded Record Type*/
```

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

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

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

The structure **tacct**, which resides with the source files of the accounting commands, represents the total accounting format used by the various accounting commands:

```
/*
 *  total accounting (for acct period), also for day
 */

struct  tacct {
   uid_t          ta_uid;        /* userid */
   char           ta_name[8];    /* login name */
   float          ta_cpu[2];     /* cum. cpu time, p/np (mins) */
   float          ta_kcore[2];   /* cum kcore-minutes, p/np */
   float          ta_con[2];     /* cum. connect time, p/np, mins */
   float          ta_du;         /* cum. disk usage */
   long           ta_pc;         /* count of processes */
   unsigned short ta_sc;         /* count of login sessions */
   unsigned short ta_dc;         /* count of disk samples */
   unsigned short ta_fee;        /* fee for special services */
};
```

SEE ALSO

acct(1M), acct(2), acctcom(1), exec(2), fork(2)

NOTES

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

NAME

admin – installation defaults file

DESCRIPTION

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

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

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

param=value

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

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

basedir Indicates the base directory where relocatable packages are to be installed. The value may contain **$PKGINST** to indicate a base directory that is to be a function of the package instance.

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

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

nocheck	Do not check for run level (system state).
quit	Abort installation if run level (system state) is not met.

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

nocheck	Do not check for conflict; files in conflict will be overwritten.
quit	Abort installation if conflict is detected.
nochange	Override installation of conflicting files; they will not be installed.

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

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

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

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

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

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

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

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

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

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

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

nocheck	Do not check package dependencies.
quit	Abort removal if package dependencies are not met.

space	Controls resolution if disk space requirements for package are not met. Options are:
	nocheck — Do not check space requirements (installation fails if it runs out of space).
	quit — Abort installation if space requirements are not met.

NOTES

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

EXAMPLES

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

SEE ALSO

pkgadd(1M)

NAME

aliases, addresses, forward – (BSD) addresses and aliases for sendmail

SYNOPSIS

/usr/ucblib/aliases
/usr/ucblib/aliases.dir
/usr/ucblib/aliases.pag
~/.forward

DESCRIPTION

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

/etc/passwd	Mail addresses (usernames) of local users.
/usr/ucblib/aliases	Aliases for the local host, in ASCII format. This file can be edited to add, update, or delete local mail aliases.
/usr/ucblib/aliases. { dir , pag}	The aliasing information from **/usr/ucblib/aliases**, in binary, **dbm** format for use by **sendmail**. The program, **newaliases**, 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** 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** pipes the message to the standard input of the *command* the bar precedes.

DARPA-standard Addresses

username@domain

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

Commercial organizations.

Educational organizations.

Government organizations.

Military organizations.

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

js@jsmachine.Podunk-U.EDU

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

uucp Addresses

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

These are sometimes mistakenly referred to as "Usenet" addresses. **uucp** 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**(1M) for details. Standard addresses are recommended.

Aliases

Local Aliases

/usr/ucblib/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 **/usr/ucblib/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 a 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 a 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** program, user **js** creates a **.forward** file that contains the line:

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

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

FILES

/etc/passwd
/usr/ucblib/aliases
~/.forward

SEE ALSO

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

NOTES

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

NAME

ar – archive file format

SYNOPSIS

#include <ar.h>

DESCRIPTION

The archive command **ar** [see **ar**(1)] is used to combine several files into one. Archives are used mainly as libraries to be searched by the link editor **ld** [see **ld**(1)].

Each archive begins with a unique string identifier called an archive magic string.

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

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

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

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

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

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

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

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

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

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

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

	0	1	2	3
0x01020304	01	02	03	04

The contents of the symbol table are as follows:

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

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

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

The 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.

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

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

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

SEE ALSO

`a.out`(4), **`ar`**(1), **`ld`**(1), **`sputl`**(3X), **`strip`**(1)

NOTES

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

NAME

archives – device header file

DESCRIPTION

```
/* Magic numbers */

#define CMN_ASC   0x070701   /* Cpio Magic Number for ASCii header */
#define CMN_BIN   070707     /* Cpio Magic Number for Binary header */
#define CMN_BBS   0143561    /* Cpio Magic Number for Byte-Swap header */
#define CMN_CRC   0x070702   /* Cpio Magic Number for CRC header */
#define CMS_ASC   "070701"   /* Cpio Magic String for ASCii header */
#define CMS_CHR   "070707"   /* Cpio Magic String for CHR (-c) header */
#define CMS_CRC   "070702"   /* Cpio Magic String for CRC header */
#define CMN_SEC   0x070703   /* Tcpio Magic Number of TI/E header */
#define CMS_SEC   "070703"   /* Tcpio Magic String of TI/E header */
#define CMS_LEN   6          /* Cpio Magic String LENgth */

/* Various header and field lengths */

#define CHRSZ     76   /* -c hdr size minus filename field */
#define ASCSZ     110  /* ASC and CRC hdr size minus filename field */
#define TARSZ     512  /* TAR hdr size */

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

/* cpio binary header definition */

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

/* cpio ODC header format */

struct c_hdr {
    char     c_magic[CMS_LEN],
             c_dev[6],
             c_ino[6],
             c_mode[6],
             c_uid[6],
             c_gid[6],
             c_nlink[6],
             c_rdev[6],
             c_mtime[11],
             c_namesz[6],
             c_filesz[11],
                                        c_name[HNAMLEN];
} ;

/* -c and CRC header format */

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

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

/* tar header definition */
union tblock {
    char dummy[TBLOCK];
    struct tar_hdr {
        char    t_name[TNAMLEN],    /* name of file */
                t_mode[TMODLEN],    /* mode of file */
                t_uid[TUIDLEN],     /* uid of file */
                t_gid[TGIDLEN],     /* gid of file */
                t_size[TSIZLEN],    /* size of file in bytes */
                t_mtime[TTIMLEN],   /* modification time of file */
                t_cksum[TCRCLEN],   /* checksum of header */
                t_typeflag,
                t_linkname[TNAMLEN],/* file this file linked with */
                t_magic[TMAGLEN],
                t_version[TVERSLEN],
                t_uname[32],
                t_gname[32],
                t_devmajor[8],
                t_devminor[8],
                t_prefix[155];
    } tbuf;
} ;
```

```
/* volcopy tape label format and structure */

#define VMAGLEN 8
#define VVOLLEN 6
#define VFILLEN 464
struct volcopy_label {
    char     v_magic[VMAGLEN],
             v_volume[VVOLLEN],
             v_reels,
             v_reel;
    long     v_time,
                 v_length,
             v_dens,
             v_reelblks,             /* u370 added field */
             v_blksize,              /* u370 added field */
             v_nblocks;              /* u370 added field */
    char     v_fill[VFILLEN];
    long     v_offset;               /* used with -e and -reel options */
    int      v_type;                 /* does tape have nblocks field? */
} ;
```

NAME

binarsys – remote system information for the **ckbinarsys** command

DESCRIPTION

binarsys contains lines of the form:

*remote_system_name***:***val*

where *val* is either **Y** or **N**. This line indicates whether that particular remote system can properly deal with messages having binary content. The absence of an entry for a particular system or absence of the **binarsys** file altogether will imply **N**o.

Blank lines or lines beginning with **#** are considered comments and ignored. Should a line of **Default=y** be encountered, the default condition for missing entries described in the previous paragraph is reversed to be **Y**es. Another line of **Default=n** will restore the default condition to **N**o.

mail is distributed with the **binarsys** file containing only a **Default=y** line.

FILES

/etc/mail/binarsys

SEE ALSO

ckbinarsys(1M), **mailsurr**(4), **mail**(1)

NAME

boot – boot options

DESCRIPTION

Options for the boot program can be set or changed with keywords in **/stand/boot**. The following are recognized by **boot**.

BOOTMSG=*string*

Change the default boot message to *string*.

MEMRANGE=*range:flag*[*,range:flag* . . .]

Tell **boot** where to look when sizing memory. A *range* is a pair of decimal addresses, separated by a dash such as **1M-4M**. The *flag* indicates how the *range* should be interpreted. The following flags are recognized:

```
256  - B_MEM_BASE (0x100)
512  - B_MEM_EXPAN (0x200)
8704 - B_MEM_FORCE (0x2000) + B_MEM_EXPAN
```

If **/stand/boot** does NOT exist, the boot program uses the CMOS values as the maximum when probing for RAM (default case).

If **/stand/boot** does exist, use the MEMRANGE entry to override the CMOS values. Examples :

Probe for the minimum of 4M or the CMOS values:

```
MEMRANGE=0-640K:256,1M-4M:512
```

Probe for 64MB and ignore the CMOS values as the maximum:

```
MEMRANGE=0-640K:256,1M-16M:512,16M-64M:8704
```

Note: if B_MEM_FORCE is set it will ignore the CMOS setting for that range. The CMOS setting can only be ignored for memory above 16MB and only if the initial address of the range is above 16MB.

In addresses, use "**M**" to indicate megabytes and "**K**" to indicate kilobytes. The first address in the pair is inclusive; the last address is exclusive. When sizing the base memory (0–640K usually) the boot code checks the CMOS for the current base memory setting. If this value is less than the current base memory value, the kernel uses this lower value instead of 640K.

variable=value

All other lines of the form *variable=value* are passed as arguments to the kernel as is, via argv[].

FILES

/stand/boot
/etc/initprog/boot

SEE ALSO

boot(1)

NAME

bootparams – boot parameter data base

SYNOPSIS

bootparams

DESCRIPTION

bootparams contains a list of client entries that diskless clients use for booting. Each entry contains the following information for each diskless client:

name   *server names and pathnames*

The first field contains the name of the diskless client. The subsequent field is a list of keys, names of servers, and pathnames.

Fields are delineated with TABs.

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

EXAMPLE

This is an example of the **bootparams** file.

```
client1 root=grpserver:/nfsroot/client1 \
   swap=grpserver:/nfsswap/client1 \
   dump=grpserver:/nfsdump/client1
```

SEE ALSO

bootparamd(1M)

compver (4)

NAME

compver – compatible versions file

DESCRIPTION

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

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

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

NOTES

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

EXAMPLE

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

```
1.3
1.0
```

SEE ALSO

depend(4)

NAME

copyright – copyright information file

DESCRIPTION

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

NAME

core – core image file

DESCRIPTION

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

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

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

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

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

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

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

COFF executable programs produce core files consisting of two parts: the first section is a copy of the system's per-user data for the process, including the general registers. The format of this section is defined in the header files **sys/user.h** and **sys/reg.h**. The remainder of a COFF core image represents the actual contents of the process data space.

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

SEE ALSO

a.out(4), **crash**(1M), **elf**(3E), **getrlimit**(2), **sdb**(1), **setuid**(2), **signal**(5)

NAME

cron, **queuedefs** – option files for **crontab** and **at**

DESCRIPTION

Options for **cron**(1M) can be set or changed with keywords in **/etc/default/cron**. The following keywords are recognized by **cron**:

CRONLOG=YES or **NO**	If **CRONLOG** is set to **YES**, all **cron** jobs are logged in **/usr/lib/cron/log**. The default is **NO**.

Options for **crontab**(1) and **at**(1) can be set or changed with keywords in **/etc/cron.d/queuedefs**. The format of the file is as follows:

```
a.4j1n
b.2j2n90w
```

The first line specifies how **at**(1) jobs are to be handled:

Start a maximum of 4 concurrent jobs per user.

Use a **nice**(1) value of 1.

Do not retry jobs that cannot start because **fork**(2) fails.

The second line specifies how **crontab**(1) jobs are to be handled:

Start a maximum of 2 concurrent jobs per user.

Use a **nice**(1) value of 2.

Wait 90 seconds, then try again to start jobs that cannot start because **fork**(2) fails.

FILES

/etc/default/cron	Control logging of **cron** jobs.
/etc/cron.d/queuedefs	Specify concurrency, priority, and retry interval.

SEE ALSO

at(1), **cron**(1M), **crontab**(1)

depend (4)

NAME

depend – software dependencies files

DESCRIPTION

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

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

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

The fields are:

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

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

pkg Indicates the package abbreviation.

name Specifies the full package name.

(arch)version Specifies a particular instance of the software. A version name cannot begin with a left parenthesis. The instance specifications, both *arch* and *version*, are completely optional but each must begin on a new line that begins with white space. If no version set is specified, any version of the indicated package will match. A version preceded by a tilde (~) indicates that any compatible version will be a match. [See **compver**(4).]

EXAMPLE

Here is a sample **depend** file (for the NFS package):

```
P  base       Base System
P  nsu        Networking Support Utilities
P  inet       Internet Utilities
P  rpc        Remote Procedure Call Utilities
P  dfs        Distributed File System Utilities
```

SEE ALSO

`compver`(4)

NAME

dfstab – file containing commands for sharing resources

DESCRIPTION

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

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

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

SEE ALSO

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

NAME

dirent – file system independent directory entry

SYNOPSIS

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

DESCRIPTION

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

The **dirent** structure is defined below.

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

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

SEE ALSO

getdents(2)

NAME

dir (cdfs) – format of CD-ROM file system (**cdfs**) directory data structure

SYNOPSIS

#include <sys/fs/iso9660.h>
#include <sys/fs/cdfs_inode.h>

DESCRIPTION

In a **cdfs** file system, the contents of a file or directory are stored in contiguous physical sectors called an extent. The contents of a directory are stored in a single extent. The contents of a file may be stored in multiple non-adjacent extents. More than one file can share the same extent. The first sector of each extent may contain an Extended Attribute Record (XAR) that describes additional attributes of the file or directory (such as the User ID, Group ID, permissions).

Each directory in a **cdfs** filesystem contains two or more Directory Records. These directory records identify the file and subdirectories owned by the directory. For each file or subdirectory in the directory, there will exist one Directory Record for each extent belonging to that file/subdirectory. Each Directory Record is of variable length and contains information such as:

the name of the file or subdirectory

the location and size of its extent

a System Use Area

Note: For a multi-extent file, there will be one directory record for each extent in the file.

Each Directory Record has a System Use Area (SUA) that stores information about other operating system standards, such as additional file-related information not defined by the ISO-9660 specification. The SUA can be used to store information required to support POSIX standards. For example, the SUA can contain the device file major/minor numbers, which are defined by the POSIX standard. The System Use Sharing Protocol (SUSP) defines how the information in the SUA is defined.

The Directory Record data structure is defined in the **iso9660.h** header file. For each **cdfs** file and directory currently being referenced, an in-core data structure, **struct cdfs_drec**, is used to store the relevant portions of all of the Directory Records belonging to that file or directory. Each **cdfs_drec** also stores other information relating to the extent and/or Directory Record. The **cdfs_drec** structure is defined in the **cdfs_inode.h** header file.

The **cdfs_drec** structure is as follows:

```
struct cdfs_drec {
    struct cdfs_drec  *drec_NextDR;       /* Pointer to next Dir Rec */
    struct cdfs_drec  *drec_PrevDR;       /* Pointer to previous Dir Rec */
    uint_t            drec_Loc;           /* Loc of media DREC (L-Sec #) */
    uint_t            drec_Offset;        /* # bytes from L-sec start */
    uint_t            drec_Len;           /* Len of media Dir Rec (Bytes) */
    uint_t            drec_XarLen;        /* Len of media XAR (Log Blk) */
    daddr_t           drec_ExtLoc;        /* Location of Extent (L-Blk #) */
    uint_t            drec_DataLen;       /* Len of File Sec data */
    timestruc_t       drec_Date;          /* Recording date and time */
    uint_t            drec_Flags;         /* Flags - See below */
    uint_t            drec_UnitSz;        /* File Unit Size */
    uint_t            drec_Interleave;    /* Interleave Gap Size */
    uint_t            drec_VolSeqNum;     /* Volume Sequence Number */
    uint_t            drec_FileIDLen;     /* Len of File ID String */
    uint_t            drec_FileIDOff;     /* Dir Rec offset of File ID */
    uint_t            drec_SysUseOff;     /* Dir Rec offset of Sys Use Area */
    uint_t            drec_SysUseSz;      /* Size of Sys Use Area */
};
```

REFERENCES

cdfs-specific **fs**(4), **cdfs**-specific **inode**(4)
System Use Sharing Protocol, and *Rock Ridge Interchange Protocol* from Rock Ridge Technical Working Group, *ISO 9660 Specification,* ISO 9660:1988(E), *Working Paper for Information Processing: Volume and File Structure of CD-ROM Information Interchange* in Optical Information Systems magazine, January/February 1987

NAME

dir (s5) – format of **s5** directories

SYNOPSIS

```
#include <sys/types.h>
#include <sys/fs/s5dir.h>
```

DESCRIPTION

A directory behaves exactly like an ordinary file, save that no user may write into a directory. The fact that a file is a directory is indicated by a bit in the mode word of its i-node entry [see the **s5**-specific **inode**(4)]. The structure of a directory entry as given in the include file is:

```
#ifndef  DIRSIZ
#define  DIRSIZ  14
#endif
struct direct
{
     o_ino_t     d_ino;      /* s5 inode type */
     char        d_name[DIRSIZ];
};
```

By convention, the first two entries in each directory are **.** for the entry itself and **..** for the parent directory. The meaning of **..** is modified for the root directory of the master file system; there is no parent, so **..** has the same meaning as **.** has.

SEE ALSO

s5_specific **inode**(4)

NAME

dir (**ufs**) – format of **ufs** directories

SYNOPSIS

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

DESCRIPTION

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

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

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

SEE ALSO

ufs-specific **fs**(4)

NAME

disk.cfg – configuration defaults for mass-storage and SCSI devices

DESCRIPTION

Default values used by the **pdiadd** and **pdirm** commands can be set or changed with keywords in **/etc/conf/pack.d/*/disk.cfg**, where the * represents the name of any PDI-capable devices supported by your release of the UNIX System.

Environment Variables

The following variables are recognized in **disk.cfg**:

NAMES Specifies the short name of this device, and is used in the UNIX System configuration as the directory name for the device, as represented by the * above. For example,

```
NAMES=adsc
```

means that the device driver will be known by the string **adsc**. The **NAMES** variable is required in **/etc/conf/pack.d/*/disk.cfg**.

NAMEL Specifies the long name of this device in the UNIX System configuration. For example,

```
NAMEL="Adaptec Host Adapter"
```

means that the device driver is an Adaptec Host Adapter. Notice that the long name must be enclosed in double quotes, because it contains space characters. This name is used for informational messages. The **NAMEL** variable is required in **/etc/conf/pack.d/*/disk.cfg**.

SHAR Specifies the value for the **ishare** flag for this device, and this flag is used in the UNIX System configuration for the device. For example,

```
SHAR=1
```

means that the device driver cannot share interrupts with other devices in your UNIX System.

IVEC Specifies the value for the interrupt vector used by this device, and is used in the UNIX System configuration for the device. For example,

```
IVEC=1
```

means that the device driver can only be configured at interrupt 1. Another way that a value for **IVEC** may be specified is

```
IVEC="14 15 11"
```

which means that the device driver can be configured at either interrupt 14, 15, or 11. This line also indicates that interrupt 14 is the default value, because it is the first value listed. The **IVEC** variable is required in **/etc/conf/pack.d/*/disk.cfg**. If the device does not use interrupts, the value 0 should be specified.

DMA1 Specifies the value for the DMA channel used by this device, and is used in the UNIX System configuration for the device. For example,

```
DMA1=1
```

means that the device driver can only be configured at DMA channel 1. Another way that a value for `DMA1` may be specified is

```
DMA1="5 6 7"
```

which means that the device driver can be configured at either DMA channel 5, 6, or 7. This line also indicates that DMA channel 5 is the default value, because it is the first value listed. The `DMA1` variable is required in `/etc/conf/pack.d/*/disk.cfg`. If the device does not use a DMA channel, the value `0` should be specified.

`IOADDR`

Specifies the value for the I/O addresses used by this device, and is used in the UNIX System configuration for the device. For example,

```
IOADDR="170-178"
```

means that the device driver can only be configured at I/O address `0x170`, and needs all addresses up to and including `0x178`. Another way that a value for `IOADDR` may be specified is

```
IOADDR="170-178 1F0-1F8"
```

which means that the device driver can be configured at starting I/O address `0x170` or `0x1F0`. The value specified after the dash always indicates the end of the address range required by this device. This line also indicates that `0x170` is the default value for starting I/O address for this device, because it is the first value listed. The `IOADDR` variable is required in `/etc/conf/pack.d/*/disk.cfg` if the device uses an address range for I/O registers. Do not use prefix `0x` when specifying values for `IOADDR`.

`MEMADDR`

Specifies the value for the memory addresses used by this device, and is used in the UNIX System configuration for the device. For example,

```
MEMADDR="C8000-C9FFF"
```

means that the device driver can only be configured at memory address `0xC8000` and needs all addresses up to and including `0xC9FFF`. Another way that a value for `MEMADDR` may be specified is

```
MEMADDR="C8000-C9FFF D6000-D7FFF"
```

which means that the device driver can be configured at starting memory address `0xC8000` or `0xD6000`. The value specified after the dash always indicates the end of the address range required by this device. This line also indicates that `0xC8000` is the default value for starting memory address for this device, because it is the first value listed. The `MEMADDR` variable is required in `/etc/conf/pack.d/*/disk.cfg`. If the device does not use an address range for a boot ROM or other purposes, a value of `0-0`" should be specified. This value is a valid value in a list of acceptable values if the use of a memory address is optional for this device. The prefix `0x` must not be used in the specification of values for `MEMADDR`.

DEVICE

Specifies the controller type for this device, and is used to control the UNIX System configuration for the device. For example,

```
DEVICE=DCD
```

means that the device is a Directly Coupled Device, while

```
DEVICE=SCSI
```

means that the device is a SCSI device. The **DEVICE** variable is required in **/etc/conf/pack.d/*/disk.cfg**. The only allowable values for **DEVICE** are **DCD** and **SCSI**.

DEVTYPE

Specifies the type of this device, and is used to control the UNIX System configuration for the device. For example,

```
DEVTYPE=DISK
```

means that the device is a disk device, while

```
DEVTYPE=TAPE
```

means that the device is a tape device. The **DEVTYPE** variable is required in **/etc/conf/pack.d/*/disk.cfg** if the value of **DEVICE** is **DCD**. The only allowable values for **DEVTYPE** are **DISK** or **TAPE**.

DCD_IPL

Must contain the same value as the default value for the **IPL** variable. This variable is used during the configuration process to record the current value of the **IPL** variable for this device. For example,

```
DCD_IPL=5
```

means that this DCD device is configured at **IPL 5** in the current UNIX System kernel. The **DCD_IPL** variable is required in **/etc/conf/pack.d/*/disk.cfg** if the value of **DEVICE** is **DCD**.

DCD_SHAR

Must contain the same value as the default value for the **SHAR** variable. This variable is used during the configuration process to record the current value of the **SHAR** variable for this device. For example,

```
DCD_SHAR=3
```

means that this DCD device is configured at **SHAR 3** in the current UNIX System kernel. The **DCD_SHAR** variable is required in **/etc/conf/pack.d/*/disk.cfg** if the value of **DEVICE** is **DCD**.

DCD_IVEC

Must contain the same value as the default value for the **IVEC** variable. This variable is used during the configuration process to record the current value of the **IVEC** variable for this device. For example,

```
DCD_IVEC=14
```

means that this DCD device is configured at **IVEC 14** in the current UNIX System kernel. The **DCD_IVEC** variable is required in **/etc/conf/pack.d/*/disk.cfg** if the value of **DEVICE** is **DCD**.

Files

`/etc/conf/pack.d/*/disk.cfg`

SEE ALSO

pdiadd(1M), **pdirm**(1M)

NAME

dump – boot dump timeout file

DESCRIPTION

The **/etc/default/dump** file contains keywords recognized by the *timeout code.* When the system boots, if there is a system dump in the swap device, the system asks if you want to **save the dump**. After *n* seconds, the system assumes that you do not. The keyword **TIME** specifies the number of seconds that the system should wait before timing out.

TIME=*n*	If *n* is zero, the **save the dump** question is never asked. If the line is missing, the system waits forever. Otherwise, the system waits *n* seconds.

Files

`/etc/default/dump`

NAME

.environ, **.pref**, **.variables** – user-preference variable files for FACE

DESCRIPTION

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

Variables found in **.environ** include:

`LOGINWIN[1-4]`	Windows that are opened when FACE is initialized
`SORTMODE`	Sort mode for file folder listings. Values include the following hexadecimal digits: `1` sorted alphabetically by name `2` files most recently modified first `800` sorted alphabetically by object type The values above may be listed in reverse order by "**OR**ing" the following value: `1000` list objects in reverse order. For example, a value of `1002` will produce a folder listing with files least recently modified displayed first. A value of `1001` would produce a "reverse" alphabetical by name listing of the folder
`DISPLAYMODE`	Display mode for file folders. Values include the following hexadecimal digits: `0` file names only `4` file names and brief description `8` file names, description, plus additional information
`WASTEPROMPT`	Prompt before emptying wastebasket (yes/no)?
`WASTEDAYS`	Number of days before emptying wastebasket
`PRINCMD[1-3]`	Print command defined to print files.
`UMASK`	Holds default permissions that files will be created with.

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

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

Variables found in **.variables** include:

`EDITOR`	Default editor
`PS1`	UNIX shell prompt

FILES

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

NAME

ethers – Ethernet address to hostname database or domain

DESCRIPTION

The **ethers** file 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 SPACE and/or TAB characters. A '**#**' indicates the beginning of a comment extending to the end of line.

The standard form for Ethernet addresses is *x*:*x*:*x*:*x*:*x*:*x* where *x* is a hexadecimal number between 0 and ff, representing one byte. The address bytes are always in network order. Host names may contain any printable character other than 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**(4) 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**(4)

NAME

fd – file descriptor files

SYNOPSIS

/dev/fd/*

DESCRIPTION

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

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

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

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

Errors

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

REFERENCES

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

NAME

`filehdr` – file header for common object file (COFF)

SYNOPSIS

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

DESCRIPTION

The common object file format (COFF) is not generated by the most recent compilers provided with UNIX System V. See **`filehdr.h`** for information about COFF header files.

See **`a.out`**(4) for information about headers generated by recent compiliers.

SEE ALSO

`a.out`(4)

NAME

fspec – format specification in text files

DESCRIPTION

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

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

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

1. a list of column numbers separated by commas, indicating tabs set at the specified columns
2. a – followed immediately by an integer *n*, indicating tabs at intervals of *n* columns
3. a – followed by the name of a "canned" tab specification

Standard tabs are specified by `t-8`, or equivalently, `t1,9,17,25,` and so on. The canned tabs that are recognized are defined by the **tabs**(1) command.

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

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

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

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

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

```
* <:t5,10,15 s72:> *
```

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

SEE ALSO

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

NAME

fstypes – file that registers distributed file system packages

DESCRIPTION

fstypes resides in directory **/etc/dfs** and lists distributed file system utilities packages installed on the system. The file system indicated in the first line of the file is the default file system. When Distributed File System (DFS) Administration commands are entered without the option **-F** *fstypes,* the system takes the file system type from the first line of the **fstypes** file.

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

SEE ALSO

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

NAME

`fs` (`bfs`) – format of the `bfs` file system volume

SYNOPSIS

```
#include <sys/types.h>
#include <sys/fs/bfs.h>
```

DESCRIPTION

The `bfs` superblock is stored on sector 0. Its format is:

```
struct bfs_bdsuphead
{
    long    bh_bfsmagic;      /* Magic number */
    off_t   bh_start;         /* Filesystem data start offset */
    off_t   bh_end;           /* Filesystem data end offset */
};

/*
 * The sanity structure is used to promote sanity in compaction.  Used
 * correctly, a crash at any point during compaction is recoverable.
 */
struct bfs_sanity
{
    daddr_t fromblock;        /* "From" block of current transfer */
    daddr_t toblock;          /* "To" block of current transfer */
    daddr_t bfromblock;       /* Backup of "from" block */
    daddr_t btoblock;         /* Backup of "to" block */
};

struct bdsuper
{
    struct bfs_bdsuphead bdsup_head;   /* Header info */
    struct bfs_sanity bdsup_sane;      /* Crash recovery info whilst
                                              compacting */
    char    bdsup_fsname[6];           /* file system name */
    char    bdsup_volume[6];           /* file system volume name */
    long    bdsup_filler[118];         /* Padding */

};
#define BFS_MAGIC    0xBADFACE         /* bfs magic number */
```

The sanity structure is used to promote sanity during compaction. It is used by **fsck**(1M) to recover from a system crash at any point during compaction.

SEE ALSO

`bfs`-specific **inode**(4)

NAME

fs (cdfs) – format of a **cdfs** file system

SYNOPSIS

```
#include <sys/fs/iso9660.h>
#include <sys/fs/cdfs_fs.h>
```

DESCRIPTION

The **cdfs** file system supports the ISO-9660 and High Sierra file system format specifications. In a **cdfs** file system, sectors 0-15 are reserved for boot information (this area is not used). The Volume Descriptor list begins at sector 16. The Volume Descriptor list contains the Primary Volume Descriptor (PVD) (which is known as the super-block in other types of file systems). Directory and file data make up the rest of the file system.

The PVD contains information such as:

The location of the root directory

The size of the file system (in logical blocks)

Various identification strings and time stamps

For each **cdfs** file system that is mounted, an in-core data structure is used to store the relevant portions of the PVD. This data structure, called the **cdfs** structure, also stores the other file system specific information. The **cdfs** structure is defined in the **cdfs_fs.h** header file. The ISO-9660 and High Sierra PVD's are defined in the **iso9660.h header file.**

The format of the **cdfs** file system structure is:

```
struct cdfs {
    uint_t             cdfs_Flags;          /* State flags for this FS */
    struct pathname    cdfs_MntPnt;         /* Pathname of mount-point */
    struct pathname    cdfs_DevNode;        /* Pathname of device node */
    struct vnode       *cdfs_DevVnode;      /* 'specfs' vnode for the device */
    struct cdfs_inode  *cdfs_RootInode;     /* Inode of CDFS root directory */
    struct cdfs_fid    cdfs_RootFid;        /* FID of Root Inode */
    enum cdfs_type     cdfs_Type;           /* File system type (9660/Hi-S) */
    daddr_t            cdfs_PvdLoc;         /* PVD location (Log Sector #) */
    uint_t             cdfs_LogSecSz;       /* Logical sector size (Bytes) */
    uint_t             cdfs_LogSecMask;     /* Convert bytes to beg of Sect */
    uint_t             cdfs_LogSecShift;    /* Convert bytes to Log Sect Num */
    uint_t             cdfs_LogBlkMask;     /* Convert bytes to beg of Blk */
    uint_t             cdfs_LogBlkShift;    /* Convert bytes to Log Blk Num */

    /*
     * Relevant PVD Information
     */
    uint_t             cdfs_LogBlkSz;       /* Logical block size (Bytes) */
    uint_t             cdfs_VolVer;         /* Version # of Vol Descr struct */
    uint_t             cdfs_FileVer;        /* Version # of Dir Rec/Path Tbl */
    uint_t             cdfs_VolSetSz;       /* Volume Set size (# of discs) */
    uint_t             cdfs_VolSeqNum;      /* Volume Sequence # (Disc #) */
    uint_t             cdfs_VolSpaceSz;     /* Volume Space Size (Bytes) */
    uint_t             cdfs_PathTabSz;      /* Path Table size (Bytes) */
    daddr_t            cdfs_PathTabLoc;     /* Path Table loc. (Log Block #) */
```

```
        timestruc_t       cdfs_CreateDate;     /* Volume Creation date/time */
        timestruc_t       cdfs_ModDate;        /* Volume Modification date/time */
        timestruc_t       cdfs_ExpireDate;     /* Volume Expiration date/time */
        timestruc_t       cdfs_EffectDate;     /* Volume Effective date/time */
        uchar_t           cdfs_VolID[32];      /* Volume ID string */
        uint_t            cdfs_RootDirOff;     /* PVD offset of Root Dir Rec */
        uint_t            cdfs_RootDirSz;      /* Size (bytes) of Root Dir Rec */

        /*
         * XCDR specific fields.
         */
        struct cd_defs    cdfs_Dflts;          /* Default IDs, perms and modes */
        uint_t            cdfs_NameConv;       /* XCDR name conversion mode */
        struct cd_uidmap  cdfs_UidMap[CD_MAXUMAP]; /* User ID map array */
        struct cd_gidmap  cdfs_GidMap[CD_MAXGMAP]; /* Group ID map array */

        /*
         * SUSP specific fields.
         */
        uint_t            cdfs_SuspSkip;       /* Value for finding SUFs in SUA */

        /*
         * RRIP specific field(s).
         */
        uint_t            cdfs_DevMap_Cnt;     /* Num of valid Device mappings*/
        struct cd_devmap  cdfs_DevMap[CD_MAXDMAP]; /* Device Node (Number) Map */
};
```

REFERENCES

cdfs-specific **dir**(4), **cdfs**-specific **inode**(4)
ISO 9660 Specification, Working paper for Information Processing: Volume and File Structure of CD-ROM Information Interchange in Optical Information Systems magazine, January/February 1987

NAME

fs (**s5**) – format of **s5** file system volume

SYNOPSIS

```
#include <sys/types.h>
#include <sys/param.h>
#include <sys/fs/s5filsys.h>
```

DESCRIPTION

Every file system storage volume has a common format for certain vital information. Every such volume is divided into a certain number of 512-byte long sectors. Sector 0 is unused and is available to contain a bootstrap program or other information.

Sector 1 is the super-block. The format of a super-block is:

```
struct    filsys
{
  ushort  s_isize;            /* size in blocks of i-list */
  daddr_t s_fsize;            /* size in blocks of entire volume */
  short   s_nfree;            /* number of addresses in s_free */
  daddr_t s_free[NICFREE];    /* free block list */
  short   s_ninode;           /* number of i-nodes in s_inode */
  o_ino_t s_inode[NICINOD];   /* free i-node list */
  char    s_flock;            /* lock during free list */
                              /* manipulation */
  char    s_ilock;            /* lock during i-list manipulation */
  char    s_fmod;             /* super block modified flag */
  char    s_ronly;            /* mounted read-only flag */
  time_t  s_time;             /* last super block update */
  short   s_dinfo[4];         /* device information */
  daddr_t s_tfree;            /* total free blocks*/
  o_ino_t s_tinode;           /* total free i-nodes */
  char    s_fname[6];         /* file system name */
  char    s_fpack[6];         /* file system pack name */
  long    s_fill[12];         /* ADJUST to make */
                              /* sizeof filsys be 512 */
  long    s_state;            /* file system state */
  long    s_magic;            /* magic number to denote new file
                              /* system */
  long    s_type;             /* type of new file system */
};

#define  FsMAGIC     0xfd187e20  /* s_magic number */

#define  Fs1b        1           /* 512-byte block */
#define  Fs2b        2           /* 1024-byte block */
#define  Fs4b        3           /* 2048-byte block */

#define  FsOKAY      0x7c269d38  /* s_state: clean */
#define  FsACTIVE    0x5e72d81a  /* s_state: active */
```

```
#define  FsBAD        0xcb096f43  /* s_state: bad root */
#define  FsBADBLK     0xbadbc14b  /* s_state: bad block */
                                  /* corrupted it */
```

s_type indicates the file system type. Currently, three types of file systems are supported: the original 512-byte logical block, the 1024-byte logical block, and the 2048-byte logical block. **s_magic** is used to distinguish the **s5** file system from other FSTypes. The **s_type** field is used to determine the blocksize of the file system; 512-bytes, 1K, or 2K. The operating system takes care of all conversions from logical block numbers to physical sector numbers.

s_state is unique for each file system and indicates the state of the file system. The numerical value of the "file system state" is computed as the sum of **s_state** and **s_time** and will ordinarily be one of **FsOKAY**, **FsACTIVE**, or **FsBAD**. A cleanly unmounted, undamaged file system is indicated by the **FsOKAY** state. After a file system had been mounted for update, the state changes to **FsACTIVE**. The state reverts to **FsOKAY** after a file system has been unmounted. A special case is used for the root file system. If it appears damaged at boot time, it is mounted but marked **FsBAD**.

s_isize is the address of the first data block after the i-list; the i-list starts just after the super-block, namely in block 2; thus the i-list is **s_isize-2** blocks long. **s_fsize** is the first block not potentially available for allocation to a file. These numbers are used by the system to check for bad block numbers; if an "impossible" block number is allocated from the free list or is freed, a diagnostic is written on the on-line console. Moreover, the free array is cleared, so as to prevent further allocation from a presumably corrupted free list.

The free list for each volume is maintained as follows. The **s_free** array contains, in **s_free[1]**, ..., **s_free[s_nfree-1]**, up to 49 numbers of free blocks. **s_free[0]** is the block number of the head of a chain of blocks constituting the free list. The first long in each free-chain block is the number (up to 50) of free-block numbers listed in the next 50 longs of this chain member. The first of these 50 blocks is the link to the next member of the chain. To allocate a block: decrement **s_nfree**, and the new block is **s_free[s_nfree]**. If the new block number is 0, there are no blocks left, so give an error. If **s_nfree** became 0, read in the block named by the new block number, replace **s_nfree** by its first word, and copy the block numbers in the next 50 longs into the **s_free** array. To free a block, check if **s_nfree** is 50; if so, copy **s_nfree** and the **s_free** array into it, write it out, and set **s_nfree** to 0. In any event set **s_free[s_nfree]** to the freed block's number and increment **s_nfree**.

s_tfree is the total free blocks available in the file system.

s_ninode is the number of free i-numbers in the **s_inode** array. To allocate an i-node: if **s_ninode** is greater than 0, decrement it and return **s_inode[s_ninode]**. If it was 0, read the i-list and place the numbers of all free i-nodes (up to 100) into the **s_inode** array, then try again. To free an i-node, provided **s_ninode** is less than 100, place its number into **s_inode[s_ninode]** and increment **s_ninode**. If **s_ninode** is already 100, do not bother to enter the freed i-node into any table. This list of i-nodes is only to speed up the allocation process; the information as to whether the i-node is really free or not is maintained in the i-node itself.

s_tinode is the total free i-nodes available in the file system.

s_flock and **s_ilock** are flags maintained in the core copy of the file system while it is mounted and their values on disk are immaterial. The value of **s_fmod** on disk is likewise immaterial; it is used as a flag to indicate that the super-block has changed and should be copied to the disk during the next periodic update of file system information.

s_ronly is a read-only flag to indicate write-protection.

s_time is the last time the super-block of the file system was changed, and is the number of seconds that have elapsed since 00:00 Jan. 1, 1970 (UTC). During a reboot, the **s_time** of the super-block for the root file system is used to set the system's idea of the time.

s_fname is the name of the file system and **s_fpack** is the name of the pack.

I-numbers begin at 1, and the storage for i-nodes begins in block 2. Also, i-nodes are 64 bytes long. I-node 1 is reserved for future use. I-node 2 is reserved for the root directory of the file system, but no other i-number has a built-in meaning. Each i-node represents one file. For the format of an i-node and its flags, see **inode**(4).

SEE ALSO

fsck(1M), **fsdb**(1M), **s5**-specific **inode**(4), **mkfs**(1M), **mount**(2)

NAME

`fs` (sfs) – format of **sfs** file system volume

SYNOPSIS

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

DESCRIPTION

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

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

```
/*
 * Super block for a file system.
 */
#define SFS_MAGIC    0xbd101155
#define UFS_MAGIC    0x011954
#define FSACTIVE    0x5e72d81a      /* fs_state: mounted */
#define FSOKAY      0x7c269d38      /* fs_state: clean */
#define FSBAD       0xcb096f43      /* fs_state: bad root */

struct  fs {
        struct  fs *fs_link;        /* linked list of file systems */
        struct  fs *fs_rlink;       /* used for incore super blocks */
        daddr_t fs_sblkno;          /* addr of super-block in filesys */
        daddr_t fs_cblkno;          /* offset of cyl-block in filesys */
        daddr_t fs_iblkno;          /* offset of inode-blocks in filesys */
        daddr_t fs_dblkno;          /* offset of first data after cg */
        long    fs_cgoffset;        /* cylinder group offset in cylinder */
        long    fs_cgmask;          /* used to calc mod fs_ntrak */
        time_t  fs_time;            /* last time written */
        long    fs_size;            /* number of blocks in fs */
        long    fs_dsize;           /* number of data blocks in fs */
        long    fs_ncg;             /* number of cylinder groups */
        long    fs_bsize;           /* size of basic blocks in fs */
        long    fs_fsize;           /* size of frag blocks in fs */
        long    fs_frag;            /* number of frags in a block in fs */
/* these are configuration parameters */
        long    fs_minfree;         /* minimum percentage of free blocks */
        long    fs_rotdelay;        /* num of ms for optimal next block */
        long    fs_rps;             /* disk revolutions per second */
/* these fields can be computed from the others */
        long    fs_bmask;           /* ''blkoff'' calc of blk offsets */
        long    fs_fmask;           /* ''fragoff'' calc of frag offsets */
        long    fs_bshift;          /* ''lblkno'' calc of logical blkno */
        long    fs_fshift;          /* ''numfrags'' calc number of frags */
/* these are configuration parameters */
        long    fs_maxcontig;       /* max number of contiguous blks */
        long    fs_maxbpg;          /* max number of blks per cyl group */
```

```
/* these fields can be computed from the others */
        long    fs_fragshift;       /* block to frag shift */
        long    fs_fsbtodb;         /* fsbtodb and dbtofsb shift constant */
        long    fs_sbsize;          /* actual size of super block */
        long    fs_csmask;          /* csum block offset */
        long    fs_csshift;         /* csum block number */
        long    fs_nindir;          /* value of NINDIR */
        long    fs_inopb;           /* value of INOPB */
        long    fs_nspf;            /* value of NSPF */
        long    fs_optim;           /* optimization preference, see below */
        long    fs_state;           /* file system state */
        long    fs_sparecon[2];     /* reserved for future constants */
/* a unique id for this filesystem (currently unused and unmaintained) */
        long    fs_id[2];           /* file system id */
/* sizes determined by number of cylinder groups and their sizes */
        daddr_t fs_csaddr;          /* blk addr of cyl grp summary area */
        long    fs_cssize;          /* size of cyl grp summary area */
        long    fs_cgsize;          /* cylinder group size */
/* these fields should be derived from the hardware */
        long    fs_ntrak;           /* tracks per cylinder */
        long    fs_nsect;           /* sectors per track */
        long    fs_spc;             /* sectors per cylinder */
/* this comes from the disk driver partitioning */
        long    fs_ncyl;            /* cylinders in file system */
/* these fields can be computed from the others */
        long    fs_cpg;             /* cylinders per group */
        long    fs_ipg;             /* inodes per group */
        long    fs_fpg;             /* blocks per group * fs_frag */
/* this data must be re-computed after crashes */
        struct  csum fs_cstotal;    /* cylinder summary information */
/* these fields are cleared at mount time */
        char    fs_fmod;            /* super block modified flag */
        char    fs_clean;           /* file system is clean flag */
        char    fs_ronly;           /* mounted read-only flag */
        char    fs_flags;           /* currently unused flag */
        char    fs_fsmnt[MAXMNTLEN]; /* name mounted on */
/* these fields retain the current block allocation info */
        long    fs_cgrotor;         /* last cg searched */
        struct  csum *fs_csp[MAXCSBUFS];/* list of fs_cs info buffers */
        long    fs_cpc;             /* cyl per cycle in postbl */
        short   fs_postbl[MAXCPG][NRPOS];/* head of blocks for each rotation */
        long    fs_magic;           /* magic number */
        u_char  fs_rotbl[1];        /* list of blocks for each rotation */
};

/*
 * Cylinder group block for a file system.
 */

#define CG_MAGIC      0x090255
struct  cg {
        struct  cg *cg_link;        /* linked list of cyl groups */
        struct  cg *cg_rlink;       /* used for incore cyl groups */
        time_t  cg_time;            /* time last written */
        long    cg_cgx;             /* we are the cgx'th cylinder group */
        short   cg_ncyl;            /* number of cyl's this cg */
        short   cg_niblk;            /* number of inode blocks this cg */
        long    cg_ndblk;            /* number of data blocks this cg */
        struct  csum cg_cs;          /* cylinder summary information */
        long    cg_rotor;            /* position of last used block */
        long    cg_frotor;           /* position of last used frag */
        long    cg_irotor;           /* position of last used inode */
        long    cg_frsum[MAXFRAG];   /* counts of available frags */
        long    cg_btot[MAXCPG];     /* block totals per cylinder */
        short   cg_b[MAXCPG][NRPOS]; /* positions of free blocks */
        char    cg_iused[MAXIPG/NBBY];/* used inode map */
        long    cg_magic;            /* magic number */
        u_char  cg_free[1];          /* free block map */
};
```

SEE ALSO

`ufs`-specific **inode**(4)

NAME

fs (ufs) – format of ufs file system volume

SYNOPSIS

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

DESCRIPTION

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

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

```
/*
 * Super block for a file system.
 */
#define FS_MAGIC     0x011954
#define FSACTIVE     0x5e72d81a      /* fs_state: mounted */
#define FSOKAY       0x7c269d38      /* fs_state: clean */
#define FSBAD        0xcb096f43      /* fs_state: bad root */

struct  fs {
        struct  fs *fs_link;         /* linked list of file systems */
        struct  fs *fs_rlink;        /* used for incore super blocks */
        daddr_t fs_sblkno;           /* addr of super-block in filesys */
        daddr_t fs_cblkno;           /* offset of cyl-block in filesys */
        daddr_t fs_iblkno;           /* offset of inode-blocks in filesys */
        daddr_t fs_dblkno;           /* offset of first data after cg */
        long    fs_cgoffset;         /* cylinder group offset in cylinder */
        long    fs_cgmask;           /* used to calc mod fs_ntrak */
        time_t  fs_time;             /* last time written */
        long    fs_size;             /* number of blocks in fs */
        long    fs_dsize;            /* number of data blocks in fs */
        long    fs_ncg;              /* number of cylinder groups */
        long    fs_bsize;            /* size of basic blocks in fs */
        long    fs_fsize;            /* size of frag blocks in fs */
        long    fs_frag;             /* number of frags in a block in fs */
/* these are configuration parameters */
        long    fs_minfree;          /* minimum percentage of free blocks */
        long    fs_rotdelay;         /* num of ms for optimal next block */
        long    fs_rps;              /* disk revolutions per second */
/* these fields can be computed from the others */
        long    fs_bmask;            /* ''blkoff'' calc of blk offsets */
        long    fs_fmask;            /* ''fragoff'' calc of frag offsets */
        long    fs_bshift;           /* ''lblkno'' calc of logical blkno */
        long    fs_fshift;           /* ''numfrags'' calc number of frags */
/* these are configuration parameters */
        long    fs_maxcontig;        /* max number of contiguous blks */
        long    fs_maxbpg;           /* max number of blks per cyl group */
```

```
/* these fields can be computed from the others */
        long    fs_fragshift;         /* block to frag shift */
        long    fs_fsbtodb;           /* fsbtodb and dbtofsb shift constant */
        long    fs_sbsize;            /* actual size of super block */
        long    fs_csmask;            /* csum block offset */
        long    fs_csshift;           /* csum block number */
        long    fs_nindir;            /* value of NINDIR */
        long    fs_inopb;             /* value of INOPB */
        long    fs_nspf;              /* value of NSPF */
        long    fs_optim;             /* optimization preference, see below */
        long    fs_state;             /* file system state */
        long    fs_sparecon[2];       /* reserved for future constants */
/* a unique id for this filesystem (currently unused and unmaintained) */
        long    fs_id[2];             /* file system id */
/* sizes determined by number of cylinder groups and their sizes */
        daddr_t fs_csaddr;            /* blk addr of cyl grp summary area */
        long    fs_cssize;            /* size of cyl grp summary area */
        long    fs_cgsize;            /* cylinder group size */
/* these fields should be derived from the hardware */
        long    fs_ntrak;             /* tracks per cylinder */
        long    fs_nsect;             /* sectors per track */
        long    fs_spc;               /* sectors per cylinder */
/* this comes from the disk driver partitioning */
        long    fs_ncyl;              /* cylinders in file system */
/* these fields can be computed from the others */
        long    fs_cpg;               /* cylinders per group */
        long    fs_ipg;               /* inodes per group */
        long    fs_fpg;               /* blocks per group * fs_frag */
/* this data must be re-computed after crashes */
        struct  csum fs_cstotal;      /* cylinder summary information */
/* these fields are cleared at mount time */
        char    fs_fmod;              /* super block modified flag */
        char    fs_clean;             /* file system is clean flag */
        char    fs_ronly;             /* mounted read-only flag */
        char    fs_flags;             /* currently unused flag */
        char    fs_fsmnt[MAXMNTLEN]; /* name mounted on */
/* these fields retain the current block allocation info */
        long    fs_cgrotor;           /* last cg searched */
        struct  csum *fs_csp[MAXCSBUFS];/* list of fs_cs info buffers */
        long    fs_cpc;               /* cyl per cycle in postbl */
        short   fs_postbl[MAXCPG][NRPOS];/* head of blocks for each rotation */
        long    fs_magic;             /* magic number */
        u_char  fs_rotbl[1];          /* list of blocks for each rotation */
};

/*
 * Cylinder group block for a file system.
 */

#define CG_MAGIC      0x090255
struct  cg {
        struct  cg *cg_link;          /* linked list of cyl groups */
        struct  cg *cg_rlink;         /* used for incore cyl groups */
        time_t  cg_time;              /* time last written */
        long    cg_cgx;               /* we are the cgx'th cylinder group */
        short   cg_ncyl;              /* number of cyl's this cg */
        short   cg_niblk;            /* number of inode blocks this cg */
        long    cg_ndblk;            /* number of data blocks this cg */
        struct  csum cg_cs;          /* cylinder summary information */
        long    cg_rotor;            /* position of last used block */
        long    cg_frotor;           /* position of last used frag */
        long    cg_irotor;           /* position of last used inode */
        long    cg_frsum[MAXFRAG];   /* counts of available frags */
        long    cg_btot[MAXCPG];     /* block totals per cylinder */
        short   cg_b[MAXCPG][NRPOS]; /* positions of free blocks */
        char    cg_iused[MAXIPG/NBBY];/* used inode map */
        long    cg_magic;            /* magic number */
        u_char  cg_free[1];          /* free block map */
};
```

SEE ALSO

ufs-specific **inode**(4)

NAME

fs (vxfs) – format of **vxfs** file system volume

SYNOPSIS

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

DESCRIPTION

The **vxfs** super-block always begins at byte offset 1024 from the start of the file system. The super-block location is fixed so utilities know where to look for it.

The super-block contains the following fundamental sizes and offsets:

fs_magic
The magic number for the file system (**VX_MAGIC**). This number identifies the file system as being a **vxfs** FSType.

fs_version
The version number of the file system layout (**VX_VERSION**), currently 1.

fs_ctime
The creation date of the file system. The **time** system call supplies the time.

fs_ectime
This field is a placeholder in instances when the creation date for a file system is expanded for more precision. It currently is zero.

fs_logstart
The block address of the first Log Area block. It currently is two.

fs_logend
The block address of the last Log Area block. The Log Area size in blocks may be specified as part of **mkfs**. If not specified, a default of 256 blocks is used. A minimum size of 32 blocks is enforced.

fs_bsize
The block size of the file system. The current choices are 1024, 2048, 4096, and 8192 bytes.

fs_size
The number of blocks in the file system, expressed as the number of blocks of size *fs_bsize*. The **fs_size** field is a signed 32 bit number. The maximum number of blocks in a **vxfs** file system is limited to 31 bits.

fs_dsize
The number of data blocks in the file system. A data block is a block which may be allocated to a file in the file system.

fs_ninode
The number of inodes in the file system. The number of inodes in the file system is subject to the following rules:

The **fs_ninode** field is a signed 32-bit number.

The number of inodes is rounded down such that the inode list in each allocation unit is an integral number of inode list blocks (see **fs_ilbsize**).

The number of inodes may be specified as part of **mkfs**. If not specified, the default will be *fs_dsize* divided by 4.

fs_nau
: The number of allocation units in the file system. The number of allocation units may be specified as part of **mkfs**.

fs_defiextsize
: The default size for indirect data extents, expressed in blocks. This field is currently unused.

fs_ilbsize
: The size of an inode list block, expressed in bytes. This size may be specified as part of **mkfs**. If not specified, a default of either 2K or *fs_bsize* (whichever is larger) is used.

fs_immedlen
: The size, in bytes, of the immediate data area in each inode. This value is 96 for the **vxfs** file system version 1.

fs_ndaddr
: The number of direct extents supported by the **VX_EXT4** mapping type (see the section describing inode list). This value is 10 for the **vxfs** file system version 1.

The preceding fields define the size and makeup of the file system. To reduce the calculations required in utilities, a number of values are derived from the fundamental values and placed in the super-block.

The super-block contains the following derived offsets:

fs_aufirst
: The address, in blocks, of the first allocation unit. There can be a gap between the end of the intent log and the first allocation unit. This gap could be used to align the first allocation unit on a desired boundary.

fs_emap
: The offset in blocks of the free extent map (emap) from the start of an allocation unit.

fs_imap
: The offset in blocks of the free inode map (imap) from the start of an allocation unit.

fs_iextop
: The offset in blocks of the extended inode operation map from the start of an allocation unit.

fs_istart
: The offset in blocks of the inode list (ilist) from the start of an allocation unit.

fs_bstart
: The offset in blocks of the first data block from the start of an allocation unit. An allocation unit header may contain padding to align the first data block.

fs_femap
: The offset in blocks of the first free extent map (emap) from the start of the file system.

fs_fimap
: The offset in blocks of the first free inode map (imap) from the start of the file system.

fs_fiextop
: The offset in blocks of the first extended inode operation map from the start of the file system.

fs_fistart
: The offset in blocks of the first ilist from the start of the file system.

fs_fbstart
: The offset in blocks of the first data block from the start of the file system.

fs_nindir
: The number of entries in an indirect address extent. An indirect address extent is currently 8192 bytes in length, making the current value for **fs_nindir** 2048.

fs_aulen
: The length of an allocation unit in blocks.

fs_auimlen
: The length of a free inode map in blocks.

fs_auemlen
: The length of a free extent map in blocks.

fs_ailen
: The length, in blocks, of the inode list for this allocation unit.

fs_aupad
: The length, in blocks, of the allocation unit alignment padding.

fs_aublocks
: The number of data blocks in an allocation unit.

fs_maxtier
: The log base 2 of **fs_aublocks**.

fs_inopb
: The number of inode entries per **fs_bsize** block in the inode list.

fs_inopau
The number of inodes in an allocation unit.

fs_inopilb
The number of inode entries per **fs_ilbsize** block in the inode list.

fs_ndiripau
Expected number of directory inodes per allocation unit.

fs_iaddrlen
The size, in blocks, of an indirect address block. An indirect address block is 8K bytes. This field will be set to (8K / *fs_bsize*).

fs_bshift
The log base 2 of *fs_bsize*. Used to convert a byte offset into a block offset.

fs_inoshift
The log base 2 of *fs_inopb*. Used to convert an inode number into a block offset in the inode list.

fs_bmask
A mask value such that (*byte_offset* + *fs_bmask*) rounds the offset to the nearest smaller block boundary.

fs_boffmask
A mask value such that (*byte_offset* + *fs_boffmask*) yields the offset from the start of the nearest smaller block boundary.

fs_inomask
A mask value such that (*inode_number* + *fs_inomask*) yields the offset from the start of the containing inode list block of the corresponding inode list entry.

fs_checksum
A simple checksum of the above fields. A macro, **VX_FSCHECKSUM** is provided to verify or calculate the checksum.

The above fields are initialized when the file system is created and do not change unless the file system is resized. These fields are replicated in each allocation unit header.

There are additional fields which are considered to be dynamic:

fs_free
The current number of free data blocks.

fs_ifree
The current number of free inodes.

fs_efree
An array of the current number of free extents of each extent size in the file system.

fs_flags
The following flags are recognized:

VX_FULLFSCK
: Set when a file system requires a full structural check to recover from an error. If this flag is set, a full check will be performed after the replay recovery is finished.

VX_NOLOG
: Set when the file system was mounted with the **VX_MS_NOLOG** option. If this flag is set, then no log replay recovery will be performed.

VX_LOGBAD
: Set when an I/O error has invalidated the log. If this flag is set, then no log replay recovery will be performed.

VX_LOGRESET
: Set when the log ID runs over **VX_MAXLOGID** (2^30). The log ID will be reset at the next appropriate opportunity (such as a mount or 60-second sync).

VX_RESIZE
: Set when a file system resizing is in progress. If an **fsck** sees this flag, it will have to perform resize recovery. Refer to **fsadm**(1M) for a description of file system expansion.

fs_mod
: Set whenever a mounted file system is modified. It is used to indicate if the super-block needs to be written when a **sync** operation is performed.

fs_clean
: Set to **VX_DIRTY** when a file system is mounted for read/write access. Set to **VX_CLEAN** upon **umount** or successful **fsck**. The **mount** utility will not allow a file system to be mounted for read/write if the **fs_clean** field is **VX_DIRTY**.

fs_reserved
: Reserved for future use.

fs_firstlogid
: Starting intent log ID to use when the file system is next mounted.

fs_time
: Last time the super-block was written to disk, indicated as the number of seconds that have elapsed since 00:00 January 1, 1970.

fs_fname
: File system name (6 characters).

fs_fpack
: File system pack label (6 characters).

SEE ALSO

fsck(1M), **fsdb**(1M), vxfs-specific **inode**(4), **mkfs**(1M), **mount**(2)

NAME

gettydefs – speed and terminal settings used by getty

DESCRIPTION

The file **/usr/lib/saf/ttymondefs** contains information used by the **getty** command to set up the speed and terminal settings for a line. It supplies information on what the **login** prompt should look like. It also supplies the speed to try next if the user indicates the current speed is not correct by typing a break character.

Each entry in **ttymondefs** has the following format:

> *label***#** *initial-flags* **#** *final-flags* **#** *login-prompt* **#***next-label*

Each entry is followed by a blank line. The fields can contain quoted characters of the form **\b**, **\n**, **\c**, etc., as well as \nnn, where *nnn* is the octal value of the desired character. The fields are:

label	This is the string against which **getty** tries to match its second argument. It is often the speed, such as **1200**, at which the terminal is supposed to run, but it need not be (see below).
initial-flags	These flags are the initial **ioctl** settings to which the terminal is to be set if a terminal type is not specified to **getty**. The flags that **getty** understands are the same as the ones listed in the **termio.h** header file [see **termio**(7)]. Normally only the speed flag is required in the *initial-flags*. **getty** automatically sets the terminal to raw input mode and takes care of most of the other flags. The *initial-flag* settings remain in effect until **getty** executes **login**.
final-flags	These flags take the same values as the *initial-flags* and are set just before **getty** executes **login**. The speed flag is again required. The composite flag **SANE** takes care of most of the other flags that need to be set so that the processor and terminal are communicating in a rational fashion. The other two commonly specified *final-flags* are **TAB3**, so that tabs are sent to the terminal as spaces, and **HUPCL**, so that the line is hung up on the final close.
login-prompt	This entire field is printed as the *login-prompt*. Unlike the above fields where white space is ignored (a space, tab or new-line), they are included in the *login-prompt* field.
next-label	If this entry does not specify the desired speed, indicated by the user typing a *break* character, then **getty** searches for the entry with *next-label* as its *label* field and sets up the terminal for those settings. Usually, a series of speeds are linked together in this fashion, into a closed set; for instance, **2400** linked to **1200**, which in turn is linked to **300**, which finally is linked to **2400**.

If **getty** is called without a second argument, then the first entry of **ttymondefs** is used, thus making the first entry of **ttymondefs** the default entry. It is also used if **getty** can not find the specified *label*. If **ttymondefs** itself is missing, there is one entry built into **getty** that brings up a terminal at **300** baud.

It is strongly recommended that after making or modifying **ttymondefs**, it be run through **getty** with the check option to be sure there are no errors.

FILES

/usr/lib/saf/ttymondefs
/usr/include/sys/termio.h

SEE ALSO

getty(1M), **ioctl**(2), **login**(1), **stty**(1), **termio**(7)

NOTES

To support terminals that pass 8 bits to the system (as is typical outside the U.S.), modify the entries in the **ttymondefs** file for those terminals as follows: add **CS8** to *initial-flags* and replace all occurrences of **SANE** with the values: **BRKINT IGNPAR ICRNL IXON OPOST ONLCR CS8 ISIG ICANON ECHO ECHOK**

An example of changing an entry in **ttymondefs** is illustrated below. All the information for an entry must be on one line in the file.

Original entry:

```
CONSOLE # B9600 HUPCL OPOST ONLCR # B9600 SANE IXANY TAB3
HUPCL # Console Login:  # console
```

Modified entry:

```
CONSOLE # B9600 CS8 HUPCL OPOST ONLCR # B9600 BRKINT IGNPAR
ICNRL IXON OPOST ONLCR CS8 ISIG ICANON ECHO ECHOK IXANY TAB3
HUPCL # Console Login:  # console
```

This change permits terminals to pass 8 bits to the system so long as the system is in multi-user state. When the system changes to single-user state, the **getty** is killed and the terminal attributes are lost. So to permit a terminal to pass 8 bits to the system in single-user state, after you are in single-user state, type [see **stty**(1)]:

```
stty -istrip cs8
```

8-bit with parity mode is not supported.

NAME

group – group file

DESCRIPTION

The file **/etc/group** contains for each group the following information:

> group name
> encrypted password
> numerical group ID
> comma-separated list of all users allowed in the group

group is an ASCII file. The fields are separated by colons; each group is separated from the next by a new-line.

Because of the encrypted passwords, the group file can and does have general read permission and can be used, for example, to map numerical group IDs to names.

During user identification and authentication, the supplementary group access list is initialized sequentially from information in this file. If a user is in more groups than the system is configured for, **{NGROUPS_MAX}**, a warning will be given and subsequent group specifications will be ignored.

SEE ALSO

getgroups(2), **groups**(1), **initgroups**(3C), **newgrp**(1M), **passwd**(1), **unistd**(4)

NAME

help – Desktop help file format

DESCRIPTION

The help subsystem can display plain help files as well as formatted help files. Formatted help files must conform to the format described below in order to take advantage of the hypertext functionality.

Headers

The file header contains information that is global to the file and must appear before the start of the first section of text. Each line in the file header begins with the ^ character. The following are the control codes and line formats allowed in the file header:

^*version^*n*

The required first line in the file header. Parse the rest of the file according to the version specified. If *n* is not specified or is set to **0**, the help file is expected to follow the syntax below. Other values of *n* are reserved.

^+*definition_file*

Specify the name of a file containing only term definitions. Each definition is defined using the ^= option defined below. Definitions can also appear in the file header itself.

^?*description*

A one-line description of the application or object the help file is being written for. This description is displayed when the icon representing the application or object in the Help Desk window is selected.

^*title^*string*

Display the title, *string*, in the help window title if no section name is found and a title is not specified in the request to the desktop manager to display help.

^%*keyword*^*reference*

Specify a keyword definition global to the file. Each occurrence of *keyword* in the file is highlighted in color. When the user selects the keyword, the text in the help window's pane area will be switched to the text which reference points to.

reference has the format *file_name*^*section_tag*. *file_name* is the name of a help file and defaults to the current file. *section_tag* is either a section name or a section tag associated with a section in the help file and, if not specified, defaults to the first section of a file. A tag is any ASCII string and can be used across different locales.

^=*term*
definition
of term
^=

Define a term global to the file. Each occurrence of *term* is displayed in italic font. When a term is selected, a window pops up to display the definition of the term. *definition* can comprise multiple lines.

Sections

A section comprises a section header followed by help text. A section header begins with a line of the format *^level^section name=alias.* Following this line must be a section tag, and, optionally, local definitions of terms and keywords, which override any previous definitions of the same terms or keywords. The same options are used to define local and global terms and keywords. The rest is considered the body of the section until the end of file is reached or another section is defined.

^level^section name[=][*alias*]

Specify section number, section name, and an optional alias to the section name. *level* starts at 0 and must be a positive integer. If *level* is 0, the section appears in the Table of Contents and help window pane without a level number. This is to allow having a main section in a file. A level 0 section is optional. A section typically starts at level 1.

section name is used internally by the desktop manager and is in the Table of Contents but not at the beginning of a section. *section name* has to be repeated on a line by itself at the beginning of the help text if you want it to appear at the start of a section.

alias is optional. If alias is specified, it is used to look up a link; otherwise, section name is used. This allows a file to have more than one section with the same name and multiple keywords that are the same but linked to different parts of the file. This is useful if section tags are not available at the time keywords are defined, which may be the case if a tool is used to create the keywords.

Note that *alias* is never displayed. In addition, the ^ after *section name* is required even if no alias is specified.

`^$`*tag* A unique section tag must be defined for each section within a section header. Since section tags are unique, they can be used to link the same keyword to different sections in a file.

Links

The following constructs are used within the help text to mark defined terms and hypertext links.

`\d(`*term*[`,`*alias*]`)`

This indicates that the string *term* is to be displayed in italics. If *alias* is not specified, the definition of *term* is displayed in a definition popup window; otherwise, the definition of *alias* is displayed in the definition window instead. *alias* allows multiple terms to share the same definition.

Definitions are specified using the ^= option in a definition file or section header.

`\k(`*keyword*[*^reference*]`)`

This is used to indicate and set up a link within a section. *keyword* appears as part of the section. When a keyword is selected, *reference* is used to look up which file and section to jump to.

reference must be in the format *file name^section tag,* of which at least file name or section tag must be specified. The section associated with *section tag* in *file name* will be displayed in the help window when *keyword* is selected.

reference allows the same keyword to be associated with the different sections in the same or a different file.

If *reference* is not specified, *keyword* will be used to look up a link, which can be a link defined in the section or file header.

Note that if `(` and `)` are used within the definition of a link, the link itself must be enclosed with curly braces instead of round parentheses and vice-versa. For example:

```
\k{cat(1)}
\k{cat^^cat(1)}
```

EXAMPLES

```
^*version^1
^*title^User Setup
^*width^70
^+help.defs
^?Manage user logins
^0^User Setup^
^$10
User Setup

The User Setup window allows you to manage who can log onto your
system by letting you add new or delete \d(logins^login) and change
the properties of existing logins.

There are three main User Setup windows:

        \k(User Setup Window)
        \k(User Setup: Properties Window [Users])
        \k(User Setup: Properties Window [Groups])
^1^User Setup Window^
^$20
\k{User Setup}
1. User Setup Window

The User Setup Window lets you maintain the \d(groups^group) defined
on your system as well as the user and system \d(logins^login).

The buttons available from the User Setup window are:

        \k(Edit Button)
        \k(View Button)
        \k(Help Button)
```

```
^2^Edit Button^
^$30
1. \k{User Setup Window}
   1.1 Edit Button
```

The Edit Button lets you add, delete, and change properties and permissions for the icons shown in the User Setup window. Note that a user cannot log onto your system until you create a \d(login) for them through the User Setup Window. Clicking SELECT on the Edit Button brings up a menu with the following options:

```
        \k(New)
        \k(Delete)
        \k(Properties)
        \k(Permissions)
^3^New^
^$40
1. \k{User Setup Window}
   1.1 \k{Edit Button}
       1.1.1 New
```

The New menu option lets you add new users to your system. Clicking SELECT on the New menu option brings up the User Setup: Add User window.

More Information:

```
        \k(User Setup: Add User Window)
^3^Delete^
^$50
1. \k{User Setup Window}
   1.1 \k{Edit Button}
       1.1.2 Delete
```

The Delete menu option lets you delete users from your system. Once a user is deleted, they cannot log onto your system.

```
^3^Properties^
^$60
1. \k{User Setup Window}
   1.1 \k{Edit Button}
       1.1.3 Properties
```

The Properties menu option lets you change the information associated with users and groups.

For user and system \d(logins^login) you can change the login name, comment, group, home directory, shell, and user ID.

For groups you can change the group name and group ID number.

Clicking SELECT on the Properties menu option brings up the User Setup: Properties window. A different window appears, depending upon whether user/system icons or group icons are in the User Setup window.

```
More information:
        \k(User Setup: Properties Window [Users]
        \k(User Setup: Properties Window [Groups]
^3^Permissions^
^$70
1. \k{User Setup Window}
   1.1 \k{Edit Button}
       1.1.4 Permissions
```

The Permissions menu option is only active when users are displayed in the User Setup window. This option allows you to make changes to the various permission categories, however, you must have the correct permissions to make changes. Clicking SELECT on the Permissions menu option brings up the Permissions window. The Permissions window allows the following to be changed:

o Owner's Administration Account

o Mount Removable Media

o Dialup Network Use

o Dialup Management

o Printer Management

o Network Management

o Package Management

o Share Remote Resources

o Share Local Resources

The Permissions window has four buttons: Apply, Reset, Cancel, and Help.

o Apply Button: The Apply button immediately applies any changes made in the Permissions window. After applying the changes, the Permissions window is closed.

o Reset Button: The Reset button reverses any changes you make in the Permissions window. The Permissions Window remains open.

o Cancel Button: The Cancel button closes the Permissions window without making any changes.

o Help Button: The Help Button provides help for the Permissions window.

```
^2^View Button^
^$80
1. \k{User Setup Window}
   1.2 View Button
```

The View Button lets you choose whether to view user icons, system icons, or group icons in the Use Setup window. Clicking SELECT on the View Button brings up a menu with the following options:

```
        \k(Users)
        \k(System)
        \k(Groups)
^3^Users^
^$90
1. \k{User Setup Window}
   1.2 \k{View Button}
       1.2.1 Users

The Users menu option lets you view user icons in the User Setup win-
dow. When you click SELECT on Users, user icons are displayed in the
User Setup window. If user icons are already displayed in the User
Setup window, then nothing happens when you click SELECT on the Users
menu item.
^3^System^
^$100
1. \k{User Setup Window}
   1.2 \k{View Button}
       1.2.2 System

The System menu option lets you view system icons in the User Setup
window. When you click SELECT on System, system icons are displayed
in the User Setup window. If system icons are already displayed in
the User Setup window, then nothing happens when you click SELECT on
the System menu item.
^3^Groups^
^$110
1. \k{User Setup Window}
   1.2 \k{View Button}
       1.2.3 Groups

The Groups menu option lets you view group icons in the User Setup
window. When you click SELECT on Groups, group icons are displayed
in the User Setup window. If group icons are already displayed in
the User Setup window, then nothing happens when you click SELECT on
the Groups menu item.
^2^Help Button^
^$120
1. \k{User Setup Window}
   1.3 Help Button

The Help button provides online help for the User Setup window.
Clicking SELECT on the Help button brings up a menu with the follow-
ing options:

        \k(User Setup^^User Setup Help)
        \k(Table of Contents)
        \k(Help Desk)
```

```
^3^User Setup^User Setup Help
^$130
1. \k{User Setup Window}
   1.3 \k{Help Button}
       1.3.1 User Setup
The User Setup menu option provides help on the User Setup window.
^3^Table of Contents^
^$140
1. \k{User Setup Window}
   1.3 \k{Help Button}
       1.3.2 Table of Contents
The Table of Contents menu option displays the list of help topics
available for the User Setup window.
^3^Help Desk^
^$150
1. \k{User Setup Window}
   1.3 \k{Help Button}
       1.3.3 Help Desk
The Help Desk menu option opens the Help Desk window. From there, you
can select the icon for the item you want to find out more about.
^1^User Setup: Properties Window [Users]^
^$160
\k{User Setup}
2. User Setup: Properties Window [Users]
The User Setup Properties window (for users) lets you change the
login name, full name (or Comment), login type (Desktop or Nondesktop
login) and other information about users.
The buttons available from the User Setup window are:
        \k(Apply Button)
        \k(Reset Button)
        \k(Cancel Button)
        \k(Help Button^^Help Button2)
^2^Apply Button^
^$170
2. \k{User Setup: Properties Window [Users]}
   2.1 Apply Button
The Apply button immediately applies any changes made in the User
Setup: Properties window. After applying the changes, the User
Setup: Properties window closes.
^2^Reset Button^
^$180
2. \k{User Setup: Properties Window [Users]}
   2.2 Reset Button
```

```
The Reset button reverses any changes you make. The User Setup Proper-
ties Window remains open.
^2^Cancel Button^
^$190
2. \k{User Setup: Properties Window [Users]}
   2.3 Cancel Button

The Cancel button closes the User Setup Properties window without mak-
ing any changes.
^2^Help Button^Help Button2
^$200
2. \k{User Setup: Properties Window [Users]}
   2.4 Help Button

The Help Button provides help for the User Setup: Properties window.
^1^User Setup: Properties Window [Groups]^
^$210
\k{User Setup}
3. User Setup: Properties Window [Groups]

The User Setup Properties window (for Groups) lets you change the
group name and group ID number.

The following \d(buttons^button) are available from the User Setup
Properties Window (Groups):

        \k(Apply Button^^Apply Button2)
        \k(Reset Button^^Reset Button2)
        \k(Cancel Button^^Cancel Button2)
        \k(Help Button^^Help Button3)
^2^Apply Button^Apply Button2
^$220
3. \k{User Setup: Properties Window [Groups]}
   3.1 Apply Button

The Apply button immediately applies any changes made in the User
Setup: Properties window. After applying the changes, the User Setup:
Properties window closes.
^2^Reset Button^Reset Button2
^$230
3. \k{User Setup: Properties Window [Groups]}
   3.2 Reset Button

The Reset button reverses any changes you make. The User Setup Proper-
ties Window remains open.
^2^Cancel Button^Cancel Button2
^$240
3. \k{User Setup: Properties Window [Groups]}
   3.3 Cancel Button

The Cancel button closes the User Setup Properties window without mak-
ing any changes
^2^Help Button^Help Button3
^$250
```

```
3. \k{User Setup: Properties Window [Groups]}
   3.4 Help Button
```

The Help Button provides help for the User Setup: Properties window.

NAME

holidays - accounting file

DESCRIPTION

holidays contains information that accounting commands use to identify prime and non-prime computing hours. This information is used to divide user cpu usage and connection time into prime and non-prime time.

All lines in the **holidays** file that begin with an asterisk (*) are comment lines. The first non-comment line of the **holidays** file must list the year, the beginning of the prime-time period, and the end of the prime-time period. Prime time is defined as time a specified interval that occurs every day, except on Saturdays, Sundays, and holidays listed in the **holidays** file. For example, the following line indicates that the file is for 1992, and that prime time starts at 8:00AM and ends at 6:00PM. Time is given in 24-hour clocktime.

```
1992 0800 1800
```

Each of the remaining non-commented lines lists one holiday. These lines must begin with the date of the holiday in mm/dd format. All remaining information on the line is ignored, so a description of the holiday can be given. The following are examples of holiday lines:

```
1/1  New Year's Day
5/25 Memorial Day
```

which is equivalent to

```
1/1
5/25
```

The **holidays** file should be updated at the end of each year. If the **holidays** file specifies a year prior to the current year, or the end of the current year is near, then the accounting commands that use the **holidays** file will issue a message stating that it needs to be updated.

FILES

/etc/acct/holidays

SEE ALSO

acctcms(1M), **acctcon**(1M), **acctprc**(1M), **runacct**(1M)

hosts (4)

NAME

hosts – host name data base

SYNOPSIS

/etc/hosts

DESCRIPTION

The **hosts** file contains information regarding the known hosts on the DARPA Internet. 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 SPACE and/or TAB characters. A '**#**' indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file. This file is normally created from the official 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 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 hosts by system and by user

DESCRIPTION

The **/etc/hosts.equiv** file contains a list of trusted hosts. When an **rlogin**(1) or **rsh**(1) request is received from a host listed in this file, and when the user making the request is listed in the **/etc/passwd** file, then the remote login is allowed with no further checking. The library routine **ruserok** will make this verification. In this case, **rlogin** does not prompt for a password, and commands submitted through **rsh** are executed. Thus, a remote user with a local user ID is said to have equivalent access from a remote host named in this file.

The format of the **hosts.equiv** file consists of a one-line entry for each host, of the form:

hostname [*username*]

The *hostname* field normally contains the name of a trusted host from which a remote login can be made, and *username* represents a single user from that host. However, an entry consisting of a single '+' indicates that all known hosts are to be trusted for all users. A host name must be the official name as listed in the **hosts**(4) database. This is the first name given in the hosts database entry; *hostname* aliases are not recognized.

The User .rhosts File

Whenever a remote login is not allowed by **hosts.equiv**, the remote login daemon checks for a **.rhosts** file in the home directory of the local login. The **.rhosts** file controls access only to the specific login where it resides.

The **.rhosts** file has the same format as the **hosts.equiv** file, but the *username* entry has a different meaning. In the **hosts.equiv** file, a *username* entry restricts remote access to the specified remote user. In the **.rhosts** file, a *username* entry changes the identity of user attempting to log in. The remote user specified by *username* can access the host as the local login and inherit the local login's permissions.

FILES

```
/etc/hosts.equiv
/etc/passwd
~/.rhosts
/etc
```

SEE ALSO

rlogin(1), **rsh**(1), **hosts**(4), **passwd**(4)

NAME

inetd.conf – Internet servers database

DESCRIPTION

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

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

Fields can be separated by either SPACE 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
: 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,
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
: **nowait** for all but single-threaded datagram servers — servers which do not release the socket until a timeout occurs (such as **comsat**(1M) and **talkd**(1M)). These must have the status **wait**. Although **tftpd**(1M) establishes separate pseudo-connections, its forking behavior can lead to a race condition unless it is also given the status **wait**.

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

server-program
: Either the pathname of a server program to be invoked by **inetd** to perform the requested service, or the value **internal** if **inetd** itself provides the service.

server-arguments
: If a server must be invoked with command-line arguments, the entire command line (including argument 0) must appear in this field (which consists of all remaining words in the entry). If the server expects **inetd** to pass it the address of its peer (for compatibility with 4.2BSD executable daemons), then the first argument to the command should be specified as '**%A**'.

FILES

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

SEE ALSO

comsat(1M), inetd(1M), rlogin(1), rsh(1), services(4), talkd(1M), tftpd(1M)

NAME

`Init` – inittab entries for a kernel module

SYNOPSIS

`Init`

DESCRIPTION

One of the kernel configuration files, an **Init** file contains information used by the **idmkinit**(1M) command to construct a module's **/etc/inittab** entry. When the **Init** component of a module's Driver Software Package (DSP) is installed, **idinstall**(1M) stores the module's **Init** file information in **/etc/conf/init.d/***module-name*, where the file *module-name* is the name of the module being installed. Package scripts should never access **/etc/conf/init.d** files directly; only the **idinstall** command should be used.

Init files contain lines consisting of one of three forms:

```
action:process
rstate:action:process
id:rstate:action:process
```

All fields are positional and must be separated by colons. Blank lines and lines beginning with '**#**' or '*****' are considered comments and are ignored.

Lines of the first form should be used for most entries. When presented with a line of this form, **idmkinit**:

Copies the *action* and *process* field values to the **inittab** entry

Generates a valid *id* field value (called a 'tag') and prepends it to the entry

Generates an *rstate* field with a value of **2**, and adds it to the entry, following the *id* field

Lines of the second form should be used when an *rstate* value other than **2** must be specified. When presented with a line of this form, **idmkinit** generates only the *id* field value and prepends it to the entry.

Lines of the third form should be used with caution. When presented with a line of this form, **idmkinit** copies the entry to the **inittab** file verbatim. It is recommended that DSPs avoid specifying lines of this form because, if more than one DSP or add-on application specifies the same *id* field value, **idmkinit** will create multiple **inittab** entries containing this *id* value. When the **/etc/init** program attempts to process the **inittab** entries with the same **id**, it will fail with an error condition.

Note that **idmkinit** determines which of the three forms is being used by searching each line for a valid *action* keyword. Valid *action* values are:

```
boot           bootwait
initdefault    off
once           ondemand
powerfail      powerwait
respawn        sysinit
wait
```

SEE ALSO

idinstall(1M), **idmkinit**(1M), **inittab**(4), **init**(1M), **System**(4)

NAME

inittab – script for **init**

DESCRIPTION

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

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

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

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

id This is one to four characters used to uniquely identify an entry.

rstate This defines the run level in which this entry is to be processed. Run-levels effectively correspond to a configuration of processes in the system. That is, each process spawned by **init** is assigned a run level or run levels in which it is allowed to exist. The run levels are represented by a number ranging from **0** through **6**. As an example, if the system is in run level **1**, only those entries having a **1** in the *rstate* field are processed. When **init** is requested to change run levels, all processes that do not have an entry in the *rstate* field for the target run level are sent the warning signal **SIGTERM** and allowed a 5-second grace period before being forcibly terminated by the kill signal **SIGKILL**. The *rstate* field can define multiple run levels for a process by selecting more than one run level in any combination from **0** through **6**. If no run level is specified, then the process is assumed to be valid at all run levels **0** through **6**. There are three other values, **a**, **b** and **c**, which can appear in the *rstate* field, even though they are not true run levels. Entries which have these characters in the *rstate* field are processed only when an **init** or **telinit** process requests them to be run (regardless of the current run level of the system). See **init**(1M). They differ from run levels in that **init** can never enter run level **a**, **b** or **c**. Also, a request for the execution of any of these processes does not change the current run level. Furthermore, a process started by an **a**, **b** or **c** command is not killed when **init** changes levels. They are killed only if their line in **inittab** is marked **off** in the *action* field, their line is deleted entirely from **inittab**, or **init** goes into single-user state.

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

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

`wait`	When **`init`** enters the run level that matches the entry's *rstate*, start the process and wait for its termination. All subsequent reads of the **`inittab`** file while **`init`** is in the same run level cause **`init`** to ignore this entry.
`once`	When **`init`** enters a run level that matches the entry's *rstate*, start the process, do not wait for its termination. When it dies, do not restart the process. If **`init`** enters a new run level and the process is still running from a previous run level change, the program is not restarted.
`boot`	The entry is to be processed the first time **`init`** goes from single-user to multi-user state after the system is booted. (If **`initdefault`** is set to **`2`**, the process runs right after the boot.) **`init`** starts the process, does not wait for its termination and, when it dies, does not restart the process.
`bootwait`	The entry is to be processed the first time **`init`** goes from single-user to multi-user state after the system is booted. (If **`initdefault`** is set to **`2`**, the process runs right after the boot.) **`init`** starts the process, waits for its termination and, when it dies, does not restart the process.
`powerfail`	Execute the process associated with this entry only when **`init`** receives a power fail signal, **`SIGPWR`** [see **`signal`**(2)].
`powerwait`	Execute the process associated with this entry only when **`init`** receives a power fail signal, **`SIGPWR`**, and wait until it terminates before continuing any processing of **`inittab`**.
`off`	If the process associated with this entry is currently running, send the warning signal **`SIGTERM`** and wait 5 seconds before forcibly terminating the process with the kill signal **`SIGKILL`**. If the process is nonexistent, ignore the entry.
`ondemand`	This instruction is really a synonym for the **`respawn`** action. It is functionally identical to **`respawn`** but is given a different keyword in order to divorce its association with run levels. This instruction is used only with the **`a`**, **`b`** or **`c`** values described in the *rstate* field.
`initdefault`	An entry with this action is scanned only when **`init`** is initially invoked. **`init`** uses this entry, if it exists, to determine which run level to enter initially. It does this by taking the highest run level specified in the *rstate* field and using that as its initial state. If the *rstate* field is empty, this is interpreted as **`0123456`** and **`init`** therefore enters run level **`6`**. This will cause the system to loop, that is, it will go to firmware and reboot continuously. Additionally, if **`init`** does not find an **`initdefault`** entry in **`inittab`**, it requests an initial run level from the user at reboot time.

sysinit Entries with this action are scanned only when **init** is initially invoked. Among other things, **sysinit** entries may be used to initialize devices on which **init** might try to ask the run level question. These entries are executed and waited for before continuing.

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

NOTICES

The **wsinit** command is required to initialize the system console. Do not remove this file, attempt to run it from the command line, or remove the line invoking it from **/etc/inittab** or **/etc/conf/init.d/kernel**.

Application code should not attempt to modify the **/etc/inittab** file during a run-level change, since the **etc/init** program ignores inittab changes then. In particular, modifying the **/etc/inittab** file while the system is shutting down will result in minor root file system damage.

FILES

/sbin/wsinit

SEE ALSO

exec(2), **init**(1M), **open**(2), **sh**(1), **signal**(2), **ttymon**(1M), **who**(1)

NAME

inode (**bfs**) – format of a **bfs** i-node

SYNOPSIS

```
#include <sys/types.h>
#include <sys/fs/bfs.h>
```

DESCRIPTION

```
struct bfs_dirent
{
        ushort   d_ino;                 /* inode number */
        daddr_t  d_sblock;              /* Start block */
        daddr_t  d_eblock;              /* End block */
        daddr_t  d_eoffset;             /* EOF disk offset (absolute) */
        struct   bfsvattr d_fattr;      /* File attributes */
};
```

For the meaning of the defined type **daddr_t** see **types**(5). The **bfsvattr** structure appears in the header file **sys/fs/bfs.h**.

SEE ALSO

bfs-specific **fs**(4), **types**(5)

NAME

inode (cdfs) – format of a **cdfs** inode

SYNOPSIS

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

DESCRIPTION

For each file and directory in a **cdfs** file system that is currently being referenced, an in-core data structure, **struct cdfs_inode**, is used to store all of the information related to that file or directory.

The information includes items such as:

the Group ID and User ID of the file or directory

the number of bytes in the file

the file or directory's permissions (read/execute)

the date and time the file or directory was created

the type of file (regular, directory, block, character, symbolic link, pipe).

The **cdfs_inode** structure is defined in the **cdfs_inode.h** header file, and is as follows:

```
struct cdfs_inode {
    struct cdfs_inode   *i_FreeFwd;      /* Free list forward link */
    struct cdfs_inode   *i_FreeBack;     /* Free list backward link */
    struct cdfs_inode   *i_HashFwd;      /* Hash list forward link */
    struct cdfs_inode   *i_HashBack;     /* Hash list backward link */
    uint_t              i_Flags;         /* Inode flags - See CDFS struct */
    struct cdfs_fid     i_Fid;           /* File ID info */
    struct cdfs_fid     i_ParentFid;     /* Parent's File ID info */
    uid_t               i_UserID;        /* User ID */
    gid_t               i_GroupID;       /* Group ID */
    uint_t              i_Mode;          /* File type, Mode, and Perms */
    uint_t              i_Size;          /* Total # of bytes in file */
    uint_t              i_LinkCnt;       /* # of links to file */
    dev_t               i_DevNum;        /* Device # of BLK/CHR file type*/
    ulong_t             i_LockOwner;     /* Process # of owner of lock */
    short               i_Count;         /* # of inode locks by lock owner */
    uint_t              i_DRcount;       /* # of Directory Records */
    struct vfs          *i_vfs;          /* File sys associated with inode */
    daddr_t             i_NextByte;      /* Next read-ahead offset (Byte) */
    int                 i_mapsz;         /* kmem_alloc'ed size */
    long                i_mapcnt;        /* mappings to file pages */
    struct cdfs_drec    *i_DirRec;       /* 1st link-list Dir Rec of file */
    struct cdfs_xar     *i_Xar;          /* XAR info from last Dir Rec */
    struct cdfs_rrip    *i_Rrip;         /* RRIP info from last Dir Rec */
    struct vnode        *i_Vnode;        /* Vnode associated with Inode */
    timestruc_t         i_AccessDate;    /* File Access date/time */
    timestruc_t         i_ModDate;       /* File Modification date/time */
    timestruc_t         i_CreateDate;    /* File Creation date/time */
    timestruc_t         i_ExpireDate;    /* File Expiration date/time */
```

```
                timestruc_t                 i_EffectDate;/* File Effective date/time */
        timestruc_t         i_AttrDate;     /* File Attribute Change date/time */
        timestruc_t         i_BackupDate;   /* File Backup date/time */
        struct pathname     i_SymLink;
        off_t               i_DirOffset;    /* Dir offset of last ref'd entry */
        ulong               i_VerCode;      /* version code attribute */
        daddr_t             i_ReadAhead;    /* File offset of read-ahead byte */
        /*
        * The following fields cause storage to be allocated for the
        * corresponding data structures.  Since each inode will usually
        * need each of these structures, this is a simple mechanism for
        * getting the needed storage.  Reference to these structures should
        * be done via the corresponding pointers allocated above.  Thus,
        * if the storage is to be dynamically allocated, very little
        * code needs to change.
        */
        struct cdfs_drec    i_DirRecStorage; /* Static storage for i_DirRec */
        struct cdfs_xar     i_XarStorage;   /* Static storage for i_Xar */
        struct cdfs_rrip    i_RripStorage;  /* Static storage for i_Rrip */
        struct vnode        i_VnodeStorage; /* Static storage for i_Vnode */
    }
```

REFERENCES

`cdfs`-specific `dir`(4), `cdfs`-specific `fs`(4)

NAME

inode (**s5**) – format of an **s5** i-node

SYNOPSIS

```
#include <sys/types.h>
#include <sys/fs/s5ino.h>
```

DESCRIPTION

An i-node for a plain file or directory in an **s5** file system has the following structure defined by **sys/fs/s5ino.h**.

```
/* Inode structure as it appears on a disk block. */

struct1dinode1      1
{
        o_mode_t      di_mode;      /* mode and type of file */
        o_nlink_t     di_nlink;     /* number of links to file */
        o_uid_t       di_uid;       /* owner's user id */
        o_gid_t       di_gid;       /* owner's group id */
        off_t         di_size;      /* number of bytes in file */
        char          di_addr[39];  /* disk block addresses */
        unsigned char di_gen;       /* file generation number */
        time_t        di_atime;     /* time last accessed */
        time_t        di_mtime;     /* time last modified */
        time_t        di_ctime;     /* time status last changed */
};

/*
 * Of the 40 address bytes:
 *      39 are used as disk addresses
 *      13 addresses of 3 bytes each
 *      and the 40th is used as a
 *  file generation number
 */
```

For the meaning of the defined types **off_t** and **time_t** see **types**(5).

SEE ALSO

s5-specific **fs**(4), **stat**(2), **l3tol**(3C), **types**(5)

NAME

inode (sfs) – format of a **sfs** inode

SYNOPSIS

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

DESCRIPTION

The inode is the focus of all local file activity in UNIX. There is a unique inode allocated for each active file, each current directory, each mounted-on file, each mapping, and the root of the file system. An inode is 'named' by its device/inumber pair. Data in **icommon** and **isecdata** (see below) is read into memory from the permanent inode on the actual volume. Data is also written to disk from the inode in memory (the incore inode) when appropriate.

The structure **inode** represents the incore inode, and contains copies of two disk inodes, whose formats are the structures **icommon** and **i_secure** (structure **i_secure** is referenced from structure inode).

The data in **icommon** and **i_secure** is common to the incore and disk inodes. Other information is also stored in the incore inode as shown below.

```
struct inode {
         /* Filesystem independent view of this inode. */
        struct inode    *i_forw;        /* hash chain, forward */
        struct inode    *i_back;        /* hash chain, back */
        struct inode    *i_freef;       /* free chain, forward */
        struct inode    *i_freeb;       /* free chain, back */
        struct vnode    *i_vp;          /* ptr to vnode */
        struct idata    *i_data;        /* pointer to the pool data      */

        /* Filesystem dependent view of this inode. */
        union  i_secure *i_secp;        /* extra memory for security data */
        struct  vnode   i_vnode;        /* vnode for this inode */

        struct      vnode       *i_devvp;       /* vnode for block I/O */
        u_short     i_flag;
        dev_t       i_dev;      /* device where inode resides */
        ino_t       i_number;   /* i number, 1-to-1 with device address */
        off_t       i_diroff;   /* offset in dir, where we
                                                found last entry */
        struct      fs *i_fs;   /* file sys associated with this inode */
        struct      dquot *i_dquot; /* quota structure controlling
                                                this file */
        short       i_owner;    /* proc index of process locking inode */
        short       i_count;    /* number of inode locks for i_owner */
        daddr_t     i_nextr;    /* next byte read offset (read-ahead) */
        ulong       i_vcode;    /* version code attribute */
        long        i_mapcnt;   /* mappings to file pages */
        int          *i_map;      /* block list for the corresponding file */
        int          i_opencnt;   /* count of opens for this inode */
        lid_t        i_diroffid;  /* last proc changing i_diroff w/o
                                                  write access */
        clock_t      i_stamp      /*time when inode was modified but not
                                  copied to the buffer cache*/
        struct       icommon i_ic;
};

struct  icommon {
        o_mode_t ic_smode;        /*  0: mode and type of file */
        short        ic_nlink;    /*  2: number of links to file */
        o_uid_t      ic_suid;     /*  4: owner's user id */
        o_gid_t      ic_sgid;     /*  6: owner's group id */
        quad         ic_size;     /*  8: number of bytes in file */
        time_t       ic_atime;    /* 16: time last accessed */
        long         ic_atspare;
        time_t       ic_mtime;    /* 24: time last modified */
        long         ic_mtspare;
        time_t       ic_ctime;    /* 32: last time inode changed */
        long         ic_ctspare;
        daddr_t      ic_db[NDADDR];/* 40: disk block addresses */
        daddr_t      ic_ib[NIADDR];/* 88: indirect blocks */
        long         ic_flags;    /* 100: status, currently unused */
        long         ic_blocks;   /* 104: blocks actually held */
        long         ic_gen;      /* 108: generation number */
        mode_t       ic_mode;     /* 112: EFT version of mode*/
        uid_t        ic_uid;      /* 116: EFT version of uid */
        gid_t        ic_gid;      /* 120: EFT version of gid */
        ulong        ic_eftflag;  /* 124: indicate EFT version*/
};

union   i_secure {
        struct icommon is_com;
        struct isecdata {
                     lid_t        isd_lid;         /* Level IDentifier */
                     long         isd_sflags;      /* flags */
                     long         isd_aclcnt;      /* ACL count */
                     long         isd_daclcnt;     /* default ACL count */
                     daddr_t      isd_aclblk;      /* extended ACL disk blk */
                     struct       acl isd_acl[NACLI];/* ACL entries */
                     lid_t        isd_cmwlid;      /* Level IDentifier for
                                                   file CMW */
                     char         isd_filler[8];   /* reserved */
        } is_secdata;
        char         is_size[128];
};
```

The structure **dinode** represents the disk inode; it is 128 bytes long and is the same as the **ufs** inode, except that there are two 128-byte inodes allocated on disk for each directory entry.

```
struct dinode {
        union {
                    struct      icommon di_icom;
                    struct      isecdata di_secdata;
                    char        di_size[128];
        } di_un;
};
```

This "alternate inode" scheme makes it look like only the even-numbered inodes on disk are used. The first inode (the even-numbered inode) is identical to the **ufs** inode, and contains all the information in the structure **icommon**.

The second inode (the "alternate", odd-numbered inode) contains the security information in the structure **isecdata**, shown below.

SEE ALSO

sfs-specific **fs**(4)

NAME

inode (ufs) – format of a ufs inode

SYNOPSIS

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

DESCRIPTION

The inode is the focus of all local file activity in UNIX. There is a unique inode allocated for each active file, each current directory, each mounted-on file, each mapping, and the root. An inode is 'named' by its dev/inumber pair. Data in icommon is read in from permanent inode on the actual volume.

```
struct inode {
/* Filesystem independent view of this inode. */
    struct inode        *i_forw;         /* hash chain, forward */
    struct inode        *i_back;         /* hash chain, back */
    struct inode        *i_freef;        /* free chain, forward */
    struct inode        *i_freeb;        /* free chain, back */
    struct vnode        *i_vp;           /* ptr to vnode */
    struct idata        *i_data;         /* pointer to the pool data */

/* Filesystem dependent view of this inode. */
    union  i_secure     *i_secp;         /* extra memory for security data */
    struct              vnode  i_vnode;  /* vnode for this inode */

    struct              vnode  *i_devvp; /* vnode for block I/O */
    u_short             i_flag;          /* inode flags (see below) */
    dev_t               i_dev;           /* device where inode resides */
    ino_t               i_number;        /* i number, 1-to-1 with device address */
    off_t               i_diroff;        /* offset in dir, where we found last entry */
    struct              fs *i_fs;        /* file sys associated with this inode */
    struct              dquot *i_dquot;  /* quota structure controlling this file */
    short               i_owner;         /* proc index of process locking inode */
    short               i_count;         /* number of inode locks for i_owner */
    daddr_t             i_nextr;         /* next byte read offset (read-ahead) */
    ulong               i_vcode;         /* version code attribute */
    long                i_mapcnt;        /* mappings to file pages */
    int                 *i_map;          /* block list for the corresponding file */
    int                 i_opencnt;       /* count of opens for this inode */
    lid_t               i_dirofflid;     /* last proc changing i_diroff w/o write acces
    clock_t             i_stamp;         /* time when inode was modified but not copied
                                          * to the buffer cache */
    struct              icommon i_ic;
};

struct  icommon {
    o_mode_t            ic_smode;        /*  0: mode and type of file */
    short               ic_nlink;        /*  2: number of links to file */
    o_uid_t             ic_suid;         /*  4: owner's user id */
    o_gid_t             ic_sgid;         /*  6: owner's group id */
    quad                ic_size;         /*  8: number of bytes in file */
#ifdef _KERNEL
```

```
                 struct timeval ic_atime;/* 16: time last accessed */
        struct timeval ic_mtime;/* 24: time last modified */
        nstruct timeval ic_ctime;/* 32: last time inode changed */
    #else
        time_t              ic_atime;       /* 16: time last accessed */
        long                ic_atspare;
        time_t              ic_mtime;       /* 24: time last modified */
        long                ic_mtspare;
        time_t              ic_ctime;       /* 32: last time inode changed */
        long                ic_ctspare;
    #endif
        daddr_t             ic_db[NDADDR];  /* 40: disk block addresses */
        daddr_t             ic_ib[NIADDR];  /* 88: indirect blocks */
        long                ic_flags;       /* 100: status, currently unused */
        long                ic_blocks;      /* 104: blocks actually held */
        long                ic_gen;         /* 108: generation number */
        mode_t              ic_mode;        /* 112: EFT version of mode*/
        uid_t               ic_uid;         /* 116: EFT version of uid */
        gid_t               ic_gid;         /* 120: EFT version of gid */
        ulong               ic_eftflag;     /* 124: indicate EFT version*/
    };
```

SEE ALSO

ufs-specific **fs**(4)

NAME

inode (vxfs) – format of a **vxfs** inode

SYNOPSIS

```
#include <sys/types.h>
#include <sys/fs/vx_inode.h>
```

DESCRIPTION

The inode list consists of *fs_inopau* inode entries in each allocation unit. An inode entry has the following format:

i_mode
: The mode and type of file.

i_nlink
: The number of links to the file.

i_uid The inode owner.

i_gid The inode group.

i_size
: The size in bytes of the file. Eight bytes have been allocated. Only four bytes are used in the first implementation.

i_atime
: Time of last access, in **timestruc_t** format.

i_mtime
: Time of last modification, in **timestruc_t** format.

i_ctime
: Time of last inode change, in **timestruc_t** format.

i_aflags
: These flags are used to control the allocation and extension of files.

VX_AF_IFBAD
: If this flag is set, the inode is invalid in some way. It should be cleared when **fsck** is run.

VX_AF_NOEXTEND
: If this flag is set, the file may not be extended once the current reservation is exceeded. The reservation may be increased by the **VX_SETEXT** ioctl, but the file will not be automatically extended.

VX_AF_NOGROW
: If this flag is set, the file may not be extended once the current reservation is exceeded. It should be cleared on truncation or when **setext** is run. This flag is usually set because an I/O error occurs while extending a file.

VX_AF_ALIGN
: If this flag is set, the file must be allocated in extents of a fixed size and alignment. If an extent of **i_fixextsize** blocks aligned on an **i_fixextsize** boundary cannot

be found, then the allocation will fail. The alignment is relative to the beginning of the allocation unit.

`i_orgtype`
Mapping type. Indicates how the inode mapping area is to be interpreted. Currently there are two mapping types supported:

`IORG_EXT4`
Mapping area consists of an array of 32-bit extent block addresses and sizes.

`IORG_IMMED`
Mapping area itself is a data block. This mapping is referred to as Immediate Inode Data.

`i_eopflags`
Extended inode operation flag area.

`i_eopdata`
Extended inode operation data area.

`i_ftarea`
This is a union. The contents are determined by file type.

For devices, the following fields are supported:

`i_rdev`
The device number of a block or character special device.

For directories, the following fields are supported:

`i_dotdot`
The parent directory inode inumber if the inode is a directory. This replaces the standard ".." entry in the first directory block. The **`vxfs`** file system does not have explicit "." and ".." entries.

For regular files, the following fields are supported:

`i_reserve`
The number of data blocks reserved for exclusive use by the file (preallocation). A preallocation may be requested using ioctl. [See **`vxfsio`**(7).]

`i_fixextsize`
Set when the inode has a fixed extent size. The default is to have a variable extent size allocation policy. A fixed extent size may be specified using ioctl. [See **`vxfsio`**(7).]

`i_blocks`
The number of blocks currently allocated to the file, including any blocks allocated for indirect address extents.

`i_gen` The generation number. A serial number which is incremented whenever the inode is freed and reallocated. It is designed to provide a "handle" for stateless servers such as NFS.

i_serial
A count of the number of times the inode metadata has been modified. This field is a 64-bit number.

ic_org
The mapping area. This field is a union based on the value of **i_orgtype** and the file system type.

For the **vxfs IORG_IMMED** organization type, the following structure is used:

i_immed
The Immediate Inode data area, **NIMMED_N** (currently 96) bytes in length (see **fs_immedlen**). Any directory or symbolic link which is <= 96 bytes in length will be stored directly in the inode.

For the **vxfs IORG_EXT4** organization type, the following structure is used:

i_spare
Four bytes of padding, not used.

i_ies Indirect extent size. This is the size in blocks of the indirect data extents in the file.

i_ie Array of indirect address extents. There are **NIADDR** (currently 2) indirect address extents. The indirect address extents are 8192 bytes long. Each indirect address extent may contain up to 2048 extent addresses. The first indirect address extent is used for single indirection. With single indirection, each entry in the indirect address extent indicates the starting block number of a data extent. The second indirect address extent is a double indirect address extent. With double indirection, each entry in the indirect address extent indicates the starting block number of a single indirect address extent.

i_dext
An array of structures containing the direct extent addresses and sizes. Up to **NDADDR_N** (currently ten) direct extents are supported. Since a variable length extent allocation policy is used, each direct extent may have a different size. Each structure contains the following elements:

i_de Direct extent address.

i_des Direct extent size.

reserved
There are 80 bytes reserved for future use.

SEE ALSO

vxfs-specific **fs**(4), **stat**(2), **types**(5)

NAME

interface – Internet network interface configuration parameters

SYNOPSIS

/etc/confnet.d/inet/interface

DESCRIPTION

The **/etc/confnet.d/inet/interface** file is used to store network interface parameters used at boot time. Each line of data in the **interface** file contains enough information to configure an IP transport provider. This information is passed on to the **slink** [see **slink**(1M)] and **ifconfig** [see **ifconfig**(1M)] programs at boot time by **/etc/confnet.d/inet/config.boot.sh**. The **/etc/confnet.d/inet/interface** file can be maintained by running **/etc/confnet.d/configure -i** [see **configure**(1M)] in interactive mode, which will call **/etc/confnet.d/inet/configure -i** [see INET-specific **configure**(1M)].

The format of the **/etc/confnet.d/inet/interface** line is a collection of colon (:) separated fields.

prefix:*unit#*:*address*:*device*:*ifconfig_opts*:*slink_opts*

Each field and its defaults (if any) are defined below.

prefix is an identifier for a driver's **netstat** [see **netstat**(1M)] statistics. Traditionally this value corresponds to the common name used for a particular device. This field can not be null and it has no default.

unit# is the index per *prefix* type in the IP internal **netstat** array, where zero is the first element's index. This field should consist of only 0-9. This field can not be null and it has no default.

address is used by **ifconfig** to initialize the transport provider. This may be the internet name from **/etc/hosts** [see **hosts**(4)] or an address in Internet standard dot notation. Null is expanded to the system nodename, obtained by searching **/etc/hosts** for the **/usr/bin/uname -n** entry. Note the system nodename should be used for only one interface line.

device is the device node name of the transport provider. It is allocated from available network devices listed in **/etc/confnet.d/netdrivers** [see **netdrivers**(4)] through the **/etc/confnet.d/configure** script. This field can not be null and it has no default.

ifconfig_opts is used to customize the **ifconfig** options used at boot time and may contain any options defined for **ifconfig** on **ifconfig**(1M). The constructed command line will take the form:

> **ifconfig** *PrefixUnit# Converted_Address ifconfig_opts* **up**
>
> where *PrefixUnit#* is the result of concatenating *prefix* and *unit#*. *PrefixUnit#* result may be used as the *interface* parameter given to **ifconfig** and **netstat**. *Converted_Address* **is the /etc/hosts** value for the *address* field. *ifconfig_opts* can be null and has no default, but it is traditionally populated with **-trailers** (needed by SVR4 transport providers). See **ifconfig**(1M) and INET-specific **configure**(1M).

A case where *ifconfig_opts* is null is when the transport provider is **lo**, (**localhost** in **/etc/hosts**). **localhost** requires no additional **ifconfig** options at boot time.

slink_opts and the **/etc/strcf** file [see **strcf**(4)] is used by **slink** to initialize the *device* into the TCP/IP protocol stack. *slink_opts* defines the strcf function and its first arguments (it is not limited to one word). *add_interface* is the default *slink_opts* value. Additional arguments will be appended to *slink_opts* to make the final form of the **slink** operation:

> *slink_opts* **ip** *device PrefixUnit*

> where *ip* will be an open file descriptor to **/dev/ip**. *device* and *PrefixUnit* are defined in the current interface entry. For a standard Ethernet board, *slink_opts* may be null; the defaults will take care of all arguments.

Files

/dev/ip
/etc/confnet.d/configure
/etc/confnet.d/inet/config.boot.sh
/etc/confnet.d/inet/configure
/etc/confnet.d/inet/interface
/etc/confnet.d/netdrivers
/etc/hosts
/usr/bin/uname

USAGE

Examples

The entry:

> lo:0:localhost:/dev/loop:netmask 0xff000000:add_loop

from **/etc/confnet.d/inet/interface** will generate the following line to be used by **slink**:

> *slink_opts* **ip** *device PrefixUnit*

The following ifconfig command would also be generated for boot time:

> ifconfig lo0 127.0.0.1 netmask 0xff000000 up

Note that the netmask arguments are present only for the purpose of this example.

REFERENCES

generic **configure**(1M), INET-specific **configure**(1M), **hosts**(4), **ifconfig**(1M), **netdrivers**(4), **netinfo**(1M), **slink**(1M), **strcf**(4)

NAME

issue – issue identification file

DESCRIPTION

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

FILES

`/etc/issue`

SEE ALSO

login(1)

NAME

lid_and_priv – distributed file system security database

DESCRIPTION

lid_and_priv is the distributed file system (DFS) security database, located in **/etc/dfs**. The **lid_and_priv** database acts as a mechanism that allows network administrators to control access to RFS and NFS resources on a server.

File entries have the format

domainname hostname level_name priv_list

where

domainname	indicates the name of an RFS client's domain. A dash (**-**) in the field indicates that the domain is the same as the server's local domain. The *domainname* field is ignored by NFS.
hostname	indicates the client's machine name.
level_name	indicates the security label, or its alias, assigned to requests from a client. A dash (**-**) in the *level_name* field indicates the default behavior.
priv_list	is a comma-separated list of privileges that the server will accept from the client. If the network administrator wants to accept the same privileges assigned to the process on the client side, then the field should contain the entry **allprivs**. See the **intro**(2) manual page for a complete list of privileges and their meanings.

The special character ***** can be used in a file entry to set up new default values. By specifying ***** in the *domainname* and *hostname* fields, the network administrator indicates that the values in the *level_name* and *priv_list* fields in that same entry are to be used as defaults, overriding the system-defined defaults.

The special character **-** is a placeholder. It can be used in a file entry in either or both of the fields *level_name* and *priv_list* to indicate that the label and/or privileges assigned to the client are the same as the defaults.

The contents of **lid_and_priv** must be loaded into the kernel whenever changes are made to the file. A network administrator loads the contents of the file into the kernel by running the **lidload**(1M) command. When **lidload** in run, all changes in the database immediately affect all NFS resources. All RFS resources are affected immediately as well, with the exception of those with open files, which are affected once the files are closed and re-opened.

SEE ALSO

intro(2), **lidload**(1M)

NOTES

It is possible for the same RFS client to have more than one entry in **lid_and_priv**, with a different domain indicated in each entry. NFS clients should have only one entry each. If an NFS client has two entries in the file, a warning message is printed and NFS acts on the information in the first entry.

NAME

`limits` – header file for implementation-specific constants

SYNOPSIS

#include <limits.h>

DESCRIPTION

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

ARG_MAX	5120	/* max length of arguments to exec */
CHAR_BIT	8	/* max # of bits in a "char" */
CHAR_MAX	127	/* max value of a "char" */
CHAR_MIN	-128	/* min value of a "char" */
CHILD_MAX	25	/* max # of processes per user id */
CLK_TCK	_sysconf(3)	/* clock ticks per second */
DBL_DIG	15	/* digits of precision of a "double" */
DBL_MAX	1.7976931348623157E+308	/* max decimal value of a "double"*/
DBL_MIN	2.2250738585072014E-308	/* min decimal value of a "double"*/
FCHR_MAX	1048576	/* max size of a file in bytes */
FLT_DIG	6	/* digits of precision of a "float" */
FLT_MAX	3.40282347e+38F	/* max decimal value of a "float" */
FLT_MIN	1.17549435E-38F	/* min decimal value of a "float" */
INT_MAX	2147483647	/* max value of an "int" */
INT_MIN	(-2147483647-1)	/* min value of an "int" */
LINK_MAX	1000	/* max # of links to a single file */
LOGNAME_MAX	8	/* max # of characters in a login name */
LONG_BIT	32	/* # of bits in a "long" */
LONG_MAX	2147483647	/* max value of a "long int" */
LONG_MIN	(-2147483647-1)	/* min value of a "long int" */
MAX_CANON	256	/* max bytes in a line for canonical processing */
MAX_INPUT	512	/* max size of a char input buffer */
MB_LEN_MAX	5	/* max # of bytes in a multibyte character */
NAME_MAX	14	/* max # of characters in a file name */
NGROUPS_MAX	16	/* max # of groups for a user */
NL_ARGMAX	9	/* max value of "digit" in calls to the NLS printf() and scanf() */
NL_LANGMAX	14	/* max # of bytes in a LANG name */
NL_MSGMAX	32767	/* max message number */
NL_NMAX	1	/* max # of bytes in N-to-1 mapping characters */
NL_SETMAX	255	/* max set number */
NL_TEXTMAX	255	/* max # of bytes in a message string */
NZERO	20	/* default process priority */
OPEN_MAX	60	/* max # of files a process can have open */
PASS_MAX	8	/* max # of characters in a password */

```
PATH_MAX     1024                       /* max # of characters in a path name */
PID_MAX      30000                      /* max value for a process ID */
PIPE_BUF     5120                       /* max # bytes atomic in write to a pipe */
PIPE_MAX     5120                       /* max # bytes written to a pipe
                                        in a write */
SCHAR_MAX    127                        /* max value of a "signed char" */
SCHAR_MIN    (-128)                     /* min value of a "signed char" */
SHRT_MAX     32767                      /* max value of a "short int" */
SHRT_MIN     (-32768)                   /* min value of a "short int" */
SSIZE_MAX    INT_MAX                    /* max value of an "int" */
STD_BLK      1024                       /* # bytes in a physical I/O block */
SYS_NMLN     257                        /* 4.0 size of utsname elements */
                                        /* also defined in sys/utsname.h */
SYSPID_MAX   1                          /* max pid of system processes */
TMP_MAX      17576                      /* max # of unique names generated
                                        by tmpnam */
UCHAR_MAX    255                        /* max value of an "unsigned char" */
UID_MAX      60000                      /* max value for a user or group ID */
UINT_MAX     4294967295                 /* max value of an "unsigned int" */
ULONG_MAX    4294967295                 /* max value of an "unsigned long int" */
USHRT_MAX    65535                      /* max value of an "unsigned short int" */
USI_MAX      4294967295                 /* max decimal value of an "unsigned" */
WORD_BIT     32                         /* # of bits in a "word" or "int" */
```

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

```
_POSIX_ARG_MAX       4096  /* max length of arguments to exec */
_POSIX_CHILD_MAX     6     /* max # of processes per user ID */
_POSIX_LINK_MAX      8     /* max # of links to a single file */
_POSIX_MAX_CANON     255   /* max # of bytes in a line of input */
_POSIX_MAX_INPUT     255   /* max # of bytes in terminal
                           input queue */
_POSIX_NAME_MAX      14    /* # of bytes in a filename */
_POSIX_NGROUPS_MAX    0    /* max # of groups in a process */
_POSIX_OPEN_MAX      16    /* max # of files a process can have open */
_POSIX_PATH_MAX      255   /* max # of characters in a pathname */
_POSIX_PIPE_BUF      512   /* max # of bytes atomic in write
                           to a pipe */
_POSIX_SSIZE_MAX     32767 /* min value stored in object of
                           type ssize_t */
_POSIX_STREAM_MAX    8     /* min number of streams (stdio)
                           that one process can have open at a time */
_POSIX_TZNAME_MAX    3     /* max number of bytes supported
                           for name of a timezone (not the TZ variable) */
```

NAME

`login` – login default file

DESCRIPTION

Options for the **`login`** program can be set or changed with keywords in **`/etc/default/login`**. The following keywords are recognized by **`login`**.

`CONSOLE`	If set, a privileged user may log in only on the terminal defined by the `CONSOLE` keyword. For example, `CONSOLE=/dev/console` means the privileged user may log in only on the integral display device attached to **`/dev/console`**. If **`CONSOLE`** is not in **`/etc/default/login`**, a privileged user may log in on any terminal.
`ALTSHELL`	If the user's shell is defined in **`/etc/passwd`** and this keyword is set to **`YES`**, the **`SHELL`** environment variable is set to the user's shell. If set to **`NO`**, the names of nonstandard shells are not put in the **`SHELL`** environment variable. The default value of this keyword is **`NO`**. For increased security, set it to **`YES`**.
`PASSREQ`	If set to **`YES`**, all users must have a password. Any user without a password is asked for one at the first opportunity permitted by the password aging set for that user. (That is, users without passwords may not change their **`NULL`** passwords if password aging is enabled for them and the minimum time before a password can be changed has not elapsed.)
`MANDPASS`	When set to **`YES`**, this keyword makes passwords mandatory for all logins (overriding **`PASSREQ`**).
`TIMEZONE`	This keyword sets the **`TZ`** variable in the environment of the user. It must match the format of the timezone set in **`/etc/TIMEZONE`**.
`HZ`	This keyword sets the environment variable **`HZ`**, the rate of the system clock, for the user logging in. The default is **`100`**.
`PATH`	This keyword sets a default path for an unprivileged user. The default is **`/usr/bin`**.
`SUPATH`	This keyword sets the default path for the privileged user logging in. Another default path for the privileged user is in **`/etc/default/su`**, which is set for privileged users who did not log in as such. The default is **`/sbin:/usr/sbin:/usr/bin:/etc`**.
`ULIMIT`	This keyword sets the maximum file size for a user. It is in units of 512-byte blocks.
`UMASK`	This keyword is the default umask for users. The default is `077`.
`IDLEWEEKS`	This keyword is the number of weeks an account may remain idle before its login is disabled.
`TIMEOUT`	This keyword is the length of time, in seconds, that **`login`** waits for a password after receiving a user name. The default is **`60`**.

MAXTRYS This keyword sets the maximum number of login attempts permitted. The default is `5`.

LOGFAILURES
This keyword sets the number of failed login attempts permitted before a record is made. The default is `5`. [See **loginlog**(4).]

DISABLETIME
This keyword sets the number of seconds to sleep after **MAXTRYS** or **LOGFAILURES** failed logins. The default is `20`.

SLEEPTIME This keyword sets the number of seconds to sleep before printing an error message. The default is `1`.

OPT_FPM This keyword is the pathname of a regular, non-executable file containing a site-specific message to a user without a password, asking that user to pick a password. The default message is **Choose one.**

DELAYEDEXIT
This keyword is used to delay the exit from the login process, for the specified number of seconds, so the user logging in has time to read messages before the screen is cleared. Its value can be set to any number between `0` and `10`; the default value is `0`. (If you try to assign a value greater than the default—`10`—the value will be set to `10`.)

FILES

`/etc/default/login`

SEE ALSO

defadm(1M), **loginlog**(4)

NAME

loginlog – log of failed login attempts

DESCRIPTION

After **LOGFAILURES** unsuccessful login attempts, where **LOGFAILURES** is 5 if not defined in **/etc/default/login**, all the attempts are logged in the file **/var/adm/loginlog**. This file contains one record for each failed attempt. Each record contains the login name, tty specification, and time.

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

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

FILES

/var/adm/loginlog
/etc/default/login

SEE ALSO

defadm(1M), **login**(1), **login**(4), **passwd**(1)

NAME

`mailcnfg` – initialization information for **`mail`** and **`rmail`**

DESCRIPTION

The **`/etc/mail/mailcnfg`** file contains initialization information for the **`mail`** and **`rmail`** commands. This file must be created initially by the administrator. Each entry in **`mailcnfg`** consists of a line of the form

Keyword = *Value*

Leading whitespace, whitespace surrounding the equal sign, and trailing whitespace is ignored. *Keyword* may not contain embedded whitespace, but whitespace may appear within *Value*. Undefined keywords or badly formed entries are silently ignored. Lines beginning with "#" are ignored.

The **`mailcnfg`** file must be world readable.

Keyword Definitions

`ADD_DATE`	If a message which originated on the local machine does not have a **`Date:`** header, and **`ADD_DATE`** has a value, one will be added.
`ADD_FROM`	If a message which originated on the local machine does not have a **`From:`** header, and **`ADD_FROM`** has a value, one will be added.
`ADD_RECEIVED`	If a message is received which has no **`Received:`** *header, and* **`ADD_RECEIVED`** *has a value, one will be added.*
`DEBUG`	Takes the same values as the **`-x`** invocation option of **`mail`**. This provides a way of setting a system-wide debug/tracing level. Typically **`DEBUG`** is set to a value of 2, which provides minimal diagnostics useful for debugging **`mail`** and **`rmail`** failures. The value of the **`-x mail`** invocation option will override any specification of **`DEBUG`** in **`mailcnfg`**.
`CLUSTER`	To identify a closely coupled set of systems by one name to all other systems, set *Value* to the cluster name. This string is used in place of the rather than the local system nodename returned by **uname**(2), such as in the surrogate file processing or for supplying the **`...remote from...`** information on the **`From`** UNIX postmark header line.
`REMOTEFROM`	This string may be set in the event that you wish to use a slightly different string in the **`...remote from...`** information on the **`From`** header line UNIX postmark header line than either the cluster name or system name.
`FAILSAFE`	In the event that the **`/var/mail`** directory is accessed via RFS or NFS within a cluster (see **`CLUSTER`** above), provisions must be made to allow for the directory not being available when local mail is to be delivered (remote system crash, RFS or NFS problems, etc.). *Value* is a string that indicates where to forward the current message for delivery. Typically this is the remote system that actually *owns* **`/var/mail`**. In this way, the message is queued for delivery to that system when

it becomes available. For example, assume a cluster of systems (**sysa**, **sysb**, **sysc**) where **/var/mail** is physically mounted on **sysc** and made available to the other machines via RFS or NFS. If **sysc** were to crash, the RFS/NFS-accessible **/var/mail** would become unavailable and local deliveries of mail would go to **/var/mail** on the local system. When **/var/mail** is re-mounted via RFS/NFS, all messages deposited in the local directory would be hidden and essentially lost. To prevent this, if **FAILSAFE** is defined in **mailcnfg**, **mail** and **rmail** check for the existence of **/var/mail/:saved**, a required subdirectory. If this subdirectory does not exist, **mail** assumes that the RFS/NFS-accessible **/var/mail** is not available and invokes the failsafe mechanism of automatically forwarding the message to *Value*. In this example *Value* would be **sysc!%n**. The %*n* keyword is expanded to be the recipient name [see **mail**(1) for details] and thus the message would be forwarded to **sysc**!*recipient_name*. Because **sysc** is not available, the message remains on the local system until **sysc** is available, and then sent there for delivery.

DEL_EMPTY_MFILE If not specified, the default action of **mail** and **rmail** is to delete empty mailfiles if the permissions are 0660 and to retain empty mailfiles if the permissions are anything else. If *Value* is **yes**, empty mailfiles are always deleted, regardless of file permissions. If *Value* is **no**, empty mailfiles are never deleted.

DOMAIN This string is used to supply the system domain name in place of the domain name returned by **sysinfo**(2).

SMARTERHOST This string may be set to a smarter host which may be referenced within the mail surrogate file via **%X**.

ARG_MAX The maximum size of the argument list and environment to be used for surrogate commands. This overrides the kernel-settable ARG_MAX parameter. On most systems, the maximum size will be 5120 bytes.

SURR_EXPORT A comma separated list of environment variables to be passed through to surrogate commands.

CNFG_EXPORT A comma separated list of mail configuration variables to be passed through to surrogate commands as environment variables.

NOCOMPILEDSURRFILE

Normally, **mail** will create a compiled version of the surrogate file, named **/etc/mail/Cmailsurr**, whenever the surrogate file or configuration file changes, and then subsequently use the compiled version. If this variable is set to any value, **mail** will ignore the compiled surrogate file.

%mailsurr_keyletter As described in **mailsurr**(4), certain pre-defined single letter keywords are textually substituted in surrogate command fields before they are executed. While none of the predefined keyletters may be changed in meaning, new ones may be defined to provide a shorthand notation for long strings (such as **/usr/lib/mail/surrcmd**) which may appear repeatedly within the **mailsurr** file. Upper case letters are reserved for future use and will be ignored if encountered here.

FILES

/etc/mail/mailcnfg
/etc/mail/mailsurr
/etc/mail/Cmailsurr
/var/mail/:saved
/usr/lib/mail/surrcmd

SEE ALSO

mail(1), **mailsurr**(4), **sysconf**(3C), **sysinfo**(2), **uname**(2)

NOTES

If **/var/mail** is accessed via RFS or NFS and the subdirectory **/var/mail/:saved** is not removed from the local system, the **FAILSAFE** mechanism will be subverted.

NAME

mailsurr – surrogate commands for routing and transport of mail

DESCRIPTION

The **mailsurr** file contains routing and transport surrogate commands used by the **mail** command. Each entry in **mailsurr** has three (or four) whitespace-separated, single quote delimited fields:

'*sender*' '*recipient*' '*command*' ['*batch*']

or a line that begins with either of

Defaults:
Translate-Defaults:

Entries and fields may span multiple lines, but leading whitespace on field continuation lines is ignored. Fields must be less than 1024 characters long after expansion (see below). Case is always ignored for making comparisons.

The sender and recipient fields are regular expressions. If the sender and recipient fields match those of the message currently being processed, the associated command is invoked.

The *command* field may have one of the following forms:

A[**ccept**]
D[**eny**] [*message*]
L[**ocal**]
T[**ranslate**] [**B=...;S=...;F=...;T=...;L=...;**] **R=**[|]*string*
< [**B=...;S=...;C=...;F=...;**] *command*
> [**B=...;W=...;**] *command*
E[**rrors**] [**B=...;W=...;**] *command*

The **mailsurr** file must be world readable. If a batching specification (**B=...;**) is given, then the *batch* field must also be given; this fourth field is discussed in the "Batching" section.

Regular Expressions

The sender and recipient fields are composed of regular expressions (REs) which are digested by the **regexp**(5) **compile** and **advance** procedures in the C library. The regular expressions matched are those from **ed**(1), with simple parentheses **()** playing the role of **\(\)** and the addition of the **+**, **?** and | operators from **egrep**(1). Any single quotes embedded within the REs must be escaped by prepending them with a backslash or the RE is not interpreted properly.

The **mail** command prepends a circumflex (^) to the start and appends a dollar sign (**$**) to the end of each RE so that it matches the entire string. Therefore it would be an error to use ^*RE*$ in the sender and recipient fields. To provide case insensitivity, all REs are converted to lower case before compilation, and all sender and recipient information is converted to lower case before comparison. This conversion is done only for the purposes of RE pattern matching; the information contained within the message's header is not modified.

The sub-expression pattern matching capabilities of **regexp** may be used in the command field, that is, \\n, where $1 \leq n \leq 9$. Any occurrences of \\n in the replacement string are themselves replaced by the corresponding parenthesized `(...)` substring in the matched pattern. The sub-expression fields from both the sender and recipient fields are accessible, with the fields numbered 1 to 9 from left to right.

Accept and Deny Commands

Accept instructs **rmail** to continue its processing with the **mailsurr** file, but to ignore any subsequent matching **Deny**. That is, unconditionally accept this message for delivery processing. **Deny** instructs **rmail** to stop processing the **mailsurr** file and to send a negative delivery notification, along with the optional message, to the originator of the message. Whichever is encountered first takes precedence.

< Delivery command

The intent of a **<** command is that it is invoked as part of the transport and delivery mechanism, with the ready-for-delivery message available to the command at its standard input. As such, there are three conditions possible when the command exits:

Success	The command successfully delivered the message. What actually constitutes successful delivery may be different within the context of different surrogates. The **rmail** process assumes that no more processing is required for the message for the current recipient.
Continue	The command performed some function (logging remote message traffic, for example) but did not do what would be considered message delivery. The **rmail** process continues to scan the **mailsurr** file looking for some other delivery mechanism.
Failure	The command encountered some catastrophic failure. The **rmail** process stops processing the message and sends to the originator of the message a non-delivery notification that includes any **stdout** and **stderr** output generated by the command.

The semantics of the **<** command field in the **mailsurr** file allow the specification of exit codes that constitute success, continue, and failure for each surrogate command individually. See the section below on "Exit State Specifications" for details.

Surrogate commands are executed by **rmail** directly. If any shell syntax is required (metacharacters, redirection, and so on), then the surrogate command must be of the form:

sh -c "*shell command line...***"**

Special care must be taken to properly escape any embedded back-slashes and other characters special to the shell as stated in the "Translate" section above.

If there are no matching **<** commands, or all matching **<** commands exit with a continue indication, **rmail** attempts to deliver the message itself by assuming that the recipient is local and delivering the message to **/var/mail/***recipient*.

Translate Command

Translate allows optional on-the-fly translation of recipient address information. The *recipient* replacement string is specified as **R=***string*.

For example, given a command line of the form

```
'.+' '([^!]+)@(.+)\.EUO\.ATT\.com' 'Translate R=attmail!\\2!\\1'
```

and a recipient address of **rob@sysa.EUO.ATT.COM** the resulting recipient address would be **attmail!sysa!rob**.

Should the first character after the equal sign be a '|', the remainder of the string is taken as a command line to be directly executed by **rmail**. If any **sh**(1) syntax is required (metacharacters, redirection, and so on), then the surrogate command must be of the form:

sh -c "*shell command line...*"

Special care must be taken to escape properly any embedded back-slashes and single or double quotes, since **rmail** uses double quoting to group whitespace delimited fields that are meant to be considered as a single argument to **execl**() [See **exec**(2)]. As stated above, any occurrences of \\n are replaced by the appropriate substring before the command is executed.

It is assumed that the executed command will write one or more replacement strings on **stdout**. The exit code of the command is examined to determine the outcome of the translation:

Success	The command successfully translated the address. If the invoked command does not return at least one replacement string (no output or just a newline), the original address is not modified. If the original address itself is returned as one of the translations, the translations are added to the list of addresses and the original address is not modified.
Failure	The command encountered some catastrophic failure. Any translations read from **stdout** will be used, but the **rmail** process will send to the originator of the message a non-delivery notification that includes any **stderr** output generated by the command.

The semantics of the **Translate** command field in the **mailsurr** file allow the specification of exit codes that constitute success and failure for each surrogate command individually. See the section below on "Exit State Specifications" for details.

The output of the translation command consists of one or more lines of output containing the new addresses, one or more per line, separated by white space. If batching is enabled (see the section on "Batching" below), the output of the translation command consists of the new addresses, but each line of output must begin with the address for which that line of output is the translation.

This mechanism is useful for mailing list expansions. For example, the command line

```
'.+'  '(.+)' 'Translate R=|findpath \\1'
```

allows local routing decisions to be made.

If the recipient address string is modified, **mailsurr** is rescanned from the beginning with the new address(es), and any prior determination of **Accept** (see above) is discarded.

When a recipient has been translated, the new recipient may again be passed through the translation command for further translation. (Recursive mailing lists are thus possible.) Some translation commands can guarantee that their output is fully translated and cannot be further translated. For these commands, the state specification `T=1;` should be given. (If the resulting address should not be processed further within the surrogate file, the state specification `L=1;` should be given. Local delivery to `/var/mail/`*recipient* will then be attempted.)

If a recipient address translates into a recipient address previously seen, it will not be placed onto the recipient list again unless the name was translated recursively back to itself.

If the returned recipient address begins with an exclamation point "!" and **`E=1;`** is given, then all leading exclamation points will be stripped. Because the default surrogate file treats leading exclamation points as special, careful consideration should be given as to whether stripping should be performed. If a recipient address is passed in to the translate command with a leading exclamation point, the exclamation point should not be stripped.

Local Command

`Local` instructs **`rmail`** to stop processing this address within the surrogate file and attempt local delivery to `/var/mail/`*recipient*. It is equivalent to a **`Translate`** command with the state specification `L=1;` and a null translation string.

Exit State Specifications

The syntax of the exit state specification is:

[*state_id*=*ec*[`,`*ec*[,. . .]]`;`][*state_id*=*ec*[,*ec*[,. . .]]`;`[. . .]]]

Whitespace may precede the exit state specification, but must follow. *state_id*s can be specified in any order. *exit_state_id* can have the value `S`, `C`, or `F`, and are used on Delivery `<` and Translation commands. The special *state_id* of `T` may be used in Translation commands, and is described in that section. The special *state_id* of `W` may be used in Postprocessing, `>` and **`Errors`**, commands, and is described in those sections. The special *state_id* of `B` may be used to specify batched processing and is described in the section "Batching".

ec can be:

any integer $0 \le n \le 255$ [Negative exit values are not possible. See **`exit`**(2) and **`wait`**(2).]

a range of integers of the form *lower_limit–upper_limit* where the limits are $\ge$ 0 and $\le$ 255, and

∗, which implies *anything*

For example, a command field of the form:

`'< S=1-5,99;C=0,12;F=*;` *command* `%R'`

indicates that exit values of 1 through 5, and 99, are to be considered success, values of 0 (zero) and 12 indicate continue, and that anything else implies failure.

It may be possible for ambiguous entries to exist if two exit states have the same value, for example, `S=12,23;C=*;F=23,52;` or `S=*;C=9;F=*;`. To account for this, **`rmail`** looks for explicit exit values (that is, not "∗") in order of success, continue, failure. Not finding an explicit match, **`rmail`** then scans for "∗" in the same order.

It is possible to eliminate an exit state completely by covering all possible values with a different default or setting a state's value to an impossible number. (Since exit values must be between 0 and 255 (inclusive), a value of 256 is a good one to use.) For example, if you had a surrogate command that was to log all message traffic, a **mailsurr** entry of

```
'(.+)' '(.+)' '<C=*; logger \\1 \\2'
```

could never indicate success or failure.

Defaults: and Translate-Defaults: Lines

If not explicitly supplied, default exit code settings are **S=0;C=*;** for **<** commands and **S=0;F=*;** for **Translate** commands. The default exit code settings for delivery and translation commands may be redefined by creating a separate line in the **mailsurr** file of one of these forms:

```
Defaults: [S=...;][C=...;][F=...;]
Translate-Defaults: [S=...;][F=...;]
```

Defaults: lines are honored and the indicated default values redefined when the line is encountered during the normal processing of the **mailsurr** file. Therefore, to redefine the defaults globally, the **Defaults:** line should be the first line in the file. It is possible to have multiple **Defaults:** lines in the **mailsurr** file, where each subsequent line overrides the previous one.

> Postdelivery command

The intent of a **>** command is that it is invoked after a successful delivery to do any post-delivery processing that may be required. Matching **>** commands are executed only if some **<** command indicates a successful delivery (see the previous section) or local delivery processing is successful. The **mailsurr** file is rescanned and all matching **>** commands, not just those following the successful **<** command, are executed in order. The exit status of an **>** command is ignored.

Some commands exit quickly, while others may take a while. The **rmail** command normally waits for the **>** command to exit before continuing its processing. If it is better not to wait for a particular command, the state specification **W=1;** should be given.

Errors command

The intent of a **Errors** command is that it is invoked after an unsuccessful delivery to do any post-delivery processing that may be required. Matching **Errors** commands are executed only if some **<** or **Translate** command indicates a failed delivery, or local delivery processing is unsuccessful. The **mailsurr** file is rescanned and all matching **Errors** commands, not just those following the failed **<** or **Translate** command, are executed in order. The exit status of an **Errors** command is ignored. The state specification **W=1;** may be used just as for the **>** command.

Batching

Some commands may be combined together for multiple recipients. For example, the delivery commands

```
uux - sysa!rmail (tony)
uux - sysa!rmail (rob)
```

may be combined together into the single delivery command

```
uux - sysa!rmail (tony) (rob)
```

Note that there are essentially two parts to each of the above commands, the non-varying *left-hand* part, and the varying *right-hand* part. Combining two delivery lines going to a common system just requires the use of the non-varying left-hand side paired with all the remaining right-hand sides. Combining the commands together allows for the more efficient delivery of mail.

Note also that there are two possible limitations to be imposed on such commands: the maximum size of the command line as limited by the UNIX system, and the maximum size of buffers used within the command. The first limitation is an administerable kernel parameter, usually 5120 bytes, for the combination of the parameters and environment (the command-line length limitation); the second limitation is a function of the command being used and varies from command to command. For example, the internal buffers of most versions of **uux** limit the command line to 1024 bytes. (Note also that there are limitations on both sides of the network; even if the limit for **uux** were raised on the local side, the limit must remain at 1024 to be portable.)

To specify that a surrogate command is to be batched, the batching specification of **B=***number* is given along with any exit code specifications. The *number* is the maximum size of the command line to be used. It may also be specified as *, in which case the system command-line limitations are used. (The system command-line limitations may be overridden by using the **mailcnfg** variable **ARG_MAX**.)

If batching is specified, then in addition to the *command* field, a fourth *batch* field must also be specified. The *command* field specifies the non-varying *left-hand* part, and the *batch* field specifies the varying *right-hand* part. For example, a specification for **uux** might be:

```
'.+' '([^!@]+)!(.+)' '< B=1024; uux – \\1!rmail' '(\\2)'
```

All surrogate commands which permit a UNIX command to be executed may be batched.

Surrogate Command Keyletter Replacement.

Certain special sequences are textually-substituted in surrogate commands before they are invoked:

%c	value of the **Content-Type:** header line if present.
%C	"**text**" or "**binary**", depending on an actual scan of the content. This is independent of the value of any **Content-Type** header line encountered (useful when calling **ckbinarsys**.)
%D	the local domain name. This will be either **DOMAIN** from **mailcnfg**, or the value returned by **sysinfo**(2).
%H	the size of the message header in bytes.
%L	the local system name. This will be either **CLUSTER** from **mailcnfg** or the value returned by **uname**.

`%l`	value of the **`Content-Length:`** header line: the size of the message body in bytes.
`%n`	the recipient's name (address).
`%O`	the recipient's original address before any translation
`%R`	the full return path to the originator (useful for sending replies, delivery failure notifications, and so on)
`%S`	the value of the **`Subject:`** header line, if present.
`%U`	the local system name, as returned by **uname**.
`%X`	the value of **SMARTERHOST** in **`mailcnfg`**.
`\\`*n*	as described above, the corresponding (. . .) substring in the matched patterns. This implies that the **`regexp`** limitation of 9 substrings is applied to the sender and recipient REs collectively.
%*keyletters*	Other lowercase keyletters as specified in **`/etc/mail/mailcnfg`**. See **`mailcnfg`**(4).

The sequences `%L`, `%U`, `%D`, and %*keyletters* are permitted within the sender and recipient fields as well as in the command fields.

Mail Surrogate Examples

Some examples of mail surrogates include the distribution of message-waiting notifications to LAN-based recipients and lighting Message-Waiting Lamps, the ability to mail output to printers, and the logging of all **`rmail`** requests between remote systems (messages passing through the local system). The following is a sample **`mailsurr`** file:

```
#
# Some common remote mail surrogates follow. To activate any
# or all of them, remove the '#' (comment indicators) from
# the beginning of the appropriate lines. Remember that they
# will be tried in the order they are encountered in the file,
# so put preferred surrogates first.

#     Prevent all shell meta-characters
'.+'  '.*[';&|^<>()].*'       'Deny'

#     Map all names of the form local-machine!user -> user
'.+'  '%L!(.+)'               'Translate R=\\1'

#     Map all names of the form uname!user -> user
#     Must be turned on when using mail in a cluster environment.
#'.+' '%U!(.+)'               'Translate R=\\1'

#     Map all names of the form user@host -> host!user
'.+'  '([^!@]+)@(.+)'         'Translate R=\\2!\\1'

#     Map all names of the form host.uucp!user -> host!user
'.+'  '([^!@]+)\\.uucp!(.+)'  'Translate R=\\1!\\2'

#     Map all names of the form host.local-domain!user -> host!user
#     DOMAIN= within /etc/mail/mailcnfg will override sysinfo(2).
'.+'  '([^!@]+)%D!(.+)'       'Translate R=\\1!\\2'
```

```
#       Allow access to 'attmail' from remote system 'sysa'
'sysa!.*'    'attmail!.+'    'Accept'

#       Deny access to 'attmail' from all other remote systems
'.+!.+'      'attmail!.+'    'Deny No access to AT&T Mail'

#       Send mail for 'laser' to attached laser printer
#       Make certain that failures are reported via return mail.
'.+'    'laser'      '< S=0;F=*; lp -dlaser'

#       Run all local names through the mail alias processor
#
'.+'    '[^!@]+'         'Translate B=*; R=|mailalias' '%n'

#       If you wish to support a user name space of user@local-domain in
#       addition to user@host.local-domain, then add the following translation,
#       where DOMAIN is the local domain, and HOST.DOMAIN is where to send the
#       mail. Note that %D contains a leading dot, so it cannot be used in the
#       first regular expression.
#'.+'   '!DOMAIN!(.+)'          'Translate R=!HOST%D!\1'

#       For remote mail via nusend
#'.+'   '([^!]+)!(.+)'    '< nusend -d \\1 -s -e -!"rmail \\2" -'

#       For remote mail via usend
'.+'    '([^!]+)!(.+)'
          '< usend -s -d\\1 -uNoLogin -!"rmail \\2" - '

#       For remote mail via uucp
'.+'    '([^!@]+)!.+'    '<S=256;C=0;
             ckbinarsys -t %C -s \\1'
'.+'    '([^!@]+)!(.+)'    '< uux - \\1!rmail (\\2)'

#       For remote mail via smtp
#'.+'   '([^!@]+)!(.+)'         '< smtpqer %R \\1 \\2'

#       If none of the above work, then let a router change the address.
#'.+'   '.*[!@].*'       'Translate R=|smail -A %n'

#       If none of the above work, then ship remote mail off to a smarter host.
#       Make certain that SMARTERHOST= is defined within /etc/mail/mailcnfg.
#'.+'   '.*[!@].*'              'Translate R=%X!%n'

#       If you have a flat name space across multiple machines, but user-names only
#       exist on disjoint machines, this entry will forward any name not known
#       locally off to the given host.
#'.+'   '[^!@]+'                'Translate T=1; R=|localmail -p -S @HOST.DOMAIN' '%n'

#       Log successful message deliveries
'(.+)' '(.+)' '> logger \\1 \\2'
```

Note that invoking **mail** to read mail does not involve the **mailsurr** file or any surrogate processing.

Security

Surrogate commands execute with the permissions of **rmail** (user ID of the invoker, group ID of **mail**). This allows surrogate commands to validate themselves, checking that their effective group ID was **mail** at invocation time. This requires that all additions to **mailsurr** be scrutinized before insertion to prevent any unauthorized access to users' mail files. (Note that some versions of the shell turn off the effective group ID. If the surrogate command is a shell script and it requires group **mail** permissions, the shell may be explicitly invoked in the surrogate command with the –p option: *sh -p shell.script*.) All surrogate commands are executed with the path **/usr/lib/mail/surrcmd:/usr/bin**, and an environment consisting of the **SHELL=/usr/bin/sh**, **HOME**, **TZ** and **LOGNAME** variables. Other environment variables may be passed by listing them in the **mailcnfg** variable **SURR_EXPORT** as a comma-separated list (e.g. **SURR_EXPORT=TERM,LINES,COLUMNS**).

Debugging New mailsurr Entries

To debug **mailsurr** files, use the **-T** option of the **mail** command. The **-T** option requires an argument that is taken as the pathname of a test **mailsurr** file. If null (as in **-T ""**), the system **mailsurr** file is used. Enter

mail -T *test_file recipient*

The result of using the **-T** option is displayed on standard output and shows the inputs and resulting transformations as **mailsurr** is processed by the **mail** command for the indicated *recipient*.

Mail messages will never be sent or delivered when using the **-T** option.

The **-d** and **-#** option may also be used to debug the system surrogate files.

FILES

/etc/mail/mailsurr	
/usr/lib/mail/surrcmd/*	surrogate commands
/etc/mail/mailcnfg	initialization information for **mail**

SEE ALSO

ckbinarsys(1M), **ed**(1), **egrep**(1), **exec**(2), **exit**(2), **mail**(1), **mail**(1), **mailalias**(1), **mailcnfg**(4), **popen**(3S), **regexp**(5), **sh**(1), **smtpqer**(1M), **sysinfo**(2), **uname**(1), **uux**(1C), **wait**(2)

NOTES

It would be unwise to install new entries into the system **mailsurr** file without verifying at least their syntactical correctness via '**mail -T** . . .' as described above.

NAME

mapchan – format of tty device mapping files

DESCRIPTION

mapchan configures the mapping of information input and output of UNIX.

Each unique channel map requires a 2048-byte buffer for mapping the input and output of characters. No buffers are required if no channels are mapped.

A method of sharing maps is implemented for channels that have the same map in place. Each additional, unique map allocates an additional buffer. The maximum number of map buffers available on a system is configured in the kernel, and is adjustable via the link kit **NEMAP** parameter. Buffers of maps no longer in use are returned for use by other maps.

EXAMPLES OF A MAP FILE

The internal character set used by UNIX System V/386 is defined by the right column of the input map, and the first column of the output map in place on that line. The default internal character set is the 8-bit ISO 8859/1 character set, which is also known as dpANS X3.4.2 and ISO/TC97/SC2. It supports the Latin alphabet and can represent most European languages.

Any character value not given is assumed to be a straight mapping. Only the differences are shown in the *mapfile* [see **mapchan**(1M)]. The left hand column must be unique. More than one occurrence of any entry is an error. Right hand column characters can appear more than once. This is many to one mapping. Nulls can be produced with compose sequences or as part of an output string.

It is recommended that no mapping be enabled on the channel used to create or modify the mapping files. This prevents any confusion of the actual values being entered due to mapping. It is also recommended that numeric rather than character representations be used in most cases, as these are not likely to be subject to mapping. Use comments to identify the characters represented. Refer to the **ascii**(5) manual page and hardware reference manual of the device being mapped for the values to assign.

```
#
#sharp/pound/cross-hatched is the comment character
#however, a quoted # ('#') is 0x23, not a comment
#
#beep, input, output, dead, compose and control
#are special keywords and should appear as shown.
#
beep                #sound bell when errors occur
input

a b
c d

dead p
q r                 # p followed by q yields r.
s t                 # p followed by s yields t.

dead u
v w                 # u followed by v yields w.
```

```
compose x          # x is the compose key (only one allowed).
y z a
B C D              # x followed by B and C yields D.

output
e f                # e is mapped to f.
g h i j            # g is mapped to hij- one to many.
k l m n o          # k is mapped lmno

control            # The control must be last

input
E 1                # The character E is followed by 1 or more
                     unmapped character

output
FG 2               # The characters FG are followed by 2 more
                     unmapped characters
```

All of the single letters above preceding the control section must be in one of these formats:

56	# decimal
045	# octal
0xfa	# hexadecimal
´b´	# quoted char
´\076´	# quoted octal
´\x4a´	# quoted hex

All of the above formats are translated to a single byte values.

The control sections (which must be the last in the file) contain specifications of character sequences which should be passed through to or from the terminal device without going through the normal **mapchan** processing. These specifications consist of two parts: a fixed sequence of one or more defined characters indicating the start of a no-map sequence, followed by a number of characters of which the actual values are unspecified.

To illustrate this, consider a cursor-control sequence which should be passed directly to the terminal without being mapped. Such a sequence would typically begin with a fixed escape sequence instructing the terminal to interpret the following two characters as a cursor position; the values of the following two characters are variable, and depend on the cursor position requested. Such a control sequence would be specified as:

```
E= 2        # Cursor control: escape = <x><y>
```

There are two subsections under the control section: the input section, which is used to filter data sent from the terminal to UNIX System V/386, and the output section, which is used to filter data sent from UNIX System V/386 to the terminal. The two fields in each control sequence are separated by white space, that is the SPACE or TAB characters. Also the # (HASH) character introduces a comment, causing the remainder of the line to be ignored. Therefore, if any of these three characters are required in the specification itself, they should be entered using one of alternative means of entering characters, as follows:

^*x*	The character produced by the terminal on pressing the CONTROL and *x* keys together.
`E` or \	The ESCAPE character, octal 033.
\c	Where *c* is one of **`b`**, **`f`**, **`l`**, **`n`**, **`r`** or **`t`**, produces BACKSPACE, FORMFEED, LINEFEED, NEWLINE, CARRIAGE RETURN or TAB characters, respectively.
0	Since the NULL character can not be represented, this sequence is not stored as the character with octal value 0200, which behaves as a NULL on most terminals.
0 or 0*n*	Specifies the octal value of the character directly.
	Followed by any any other character is interpreted as that character. This can be used to enter SPACE, TAB, or HASH characters.

DIAGNOSTICS

`mapchan` performs these error checks when processing the *mapfile*:

more than one compose key
characters mapped to more than one thing
syntax errors in the byte values
missing input or output keywords
dead or compose keys also occurring in the input section
extra information on a line
mapping a character to null
starting an output control sequence with a character that is already mapped

If characters are displayed as the 7-bit value instead of the 8-bit value, use **`stty -a`** [see **`stty`**(1)] to verify that **`-strip`** is set. Make sure **`input`** is mapping to the 8859 character set. Dead and compose sequences are **`input`** mapping and should be going to 8859.

FILES

`/etc/default/mapchan`
`/usr/lib/mapchan/*`

NOTES

Some non-U.S keyboards and display devices do not support characters commonly used by UNIX command shells and the C programming language. Do not attempt to use such devices for system administration tasks.

Not all terminals or printers can display all the characters that can be represented using this utility. Refer to the device's hardware manual for information on the capabilities of the peripheral device.

Use of mapping files that specify a different internal character set per-channel, or a set other than the 8-bit ISO 8859 set supplied by default can cause strange side effects. It is especially important to retain the 7-bit ASCII portion of the character set [see **`ascii`**(5)] UNIX System V/386 utilities and applications assume these values. Media transported between machines with different internal code set mappings may not be portable as no mapping is performed on block devices, such as tape and floppy drives. **`trchan`** can be used to translate from one internal character set to another.

Do not set **ISTRIP** [see **stty**(1)] on channels that have mapping that includes eight bit character set.

SEE ALSO

ascii(5), **keyboard**(7), **lp**(7), **mapchan**(1M), **mapkey**(1M), **stty**(1), **trchan**(1), **tty**(7)

NAME

Master – generic configuration information for a kernel module

SYNOPSIS

Master

DESCRIPTION

One of the ID/TP kernel configuration files, a **Master** file describes a kernel module that can potentially be configured into the system. Configuration information for the individual kernel modules that are actually to be included in the next system to be built is described in the **System** file [see **System**(4)].

When the **Master** component of a module's Driver Software Package (DSP) is installed, **idinstall**(1M) stores the module's **Master** file information in **/etc/conf/mdevice.d/***module-name*, where the file *module-name* is the name of the module being installed. Package scripts should never access **/etc/conf/mdevice.d** files directly; only the **idinstall** and **idcheck**(1M) commands should be used.

Master files contain lines of the form:

> **$version** *version-number*
> **$entry** *entry-point-list*
> **$depend** *module-name-list*
> **$modtype** *loadable-module-type-name*
> *module-name prefix characteristics order bmaj cmaj*

Blank lines and lines beginning with '**#**' or '*****' are considered comments and are ignored.

The first four types of lines are as follows:

$version If present in the file, this line must appear as the first non-comment line. The line specifies the version number of the **Master** file format. The **Master** file format being described here is version 1. If this line is omitted, version 0 (the old **Master** file format) is assumed.

$entry Specifies the names of the entry point routines included in the module. One or more **$entry** lines may be used to specify the entry point names. If a single **$entry** line specifies more than one entry point name, the multiple names must be separated by white space.

The function names are constructed by appending the entry point name to the module's prefix. Only functions explicitly listed on **$entry** lines will be called directly by the kernel. The following entry points are supported (note that some of the supported entry points apply only to certain module types):

```
_init chpoll close core exec halt init intr ioctl
kenter kexit mmap msgio open poll print read segmap
size start strategy write
```

$depend	Used for dynamically loadable kernel modules only, the line specifies the names of the loadable modules (if any) that contain symbols referenced by this loadable module. One or more **$depend** lines may be used to specify the module names. If a single **$depend** line specifies more than one module name, the multiple names must be separated by white space.
$modtype	This line is used for dynamically loadable kernel modules only. The line specifies the character string (maximum of 40 characters, including white space characters) to be used to identify the type of this module in error messages.

The last line of the **Master** file contains the following six fields. Each field must be separated by white space and specify a value.

module-name Specifies the internal name of the module (maximum of 14 characters). The first character must be alphabetic; the remaining characters may be letters, digits or underscores.

prefix Specifies the character string prepended to all entry-point routines and variable names associated with this module (maximum of 8 characters). During the kernel build process, an all uppercase version of this string will also be used to construct the **#define** symbolic constants accessible to the module's **Space.c** file [see **idbuild**(1M)].

If the module has no entry-points or special variables, this field may contain a dash; in this case, no **#define** symbols will be generated.

characteristics Defines a set of flags that indicate the characteristics of the module. If none of the characteristics listed below apply to the module, the *characteristics* field must contain a dash. Valid field values are:

b	The module is a 'block' device driver.
c	The module is a (STREAMS or non-STREAMS) character device driver.
d	The module is a dispatcher class module.
e	The module is an exec object-specific module.
h	The module controls hardware, but is not a device driver; that is, the module requires hardware I/O resources (for example, interrupts or bus addresses), but does not require switch table entries.
k	Keep majors flag. This flag is intended for device drivers supplied with the base system only. It indicates that **idinstall** should use the major numbers specified by the *bmaj* and/or *cmaj* fields in the module's **Master** file, instead of automatically assigning major numbers to the module. These reserved major numbers must also be specified in the **res_major** file [see **res_major(4)**].

`m`	The module is a STREAMS module.
`o`	The module may have only one **`System`** file entry.
`r`	The module is required in all configurations of the kernel. This flag is intended for device drivers supplied with the base system only. Note that, once made, a required module's device nodes (special files in the **`/dev`** directory) are not removed [see **`idmknod`**(1M)].
`t`	The module is a non-STREAMS tty driver.
`u`	The module is a device driver which requires identical block major numbers and character major numbers. Note that both the **`b`** and **`c`** flags must also be set when using this flag; if they are not set, the **`u`** flag is ignored.
`D`	The module is a hardware module which can share its DMA channel(s) with other drivers.
`F`	The module is a VFS file system module.
`O`	The IOA range of this device may overlap that of another device.
`S`	The module is a STREAMS driver and/or STREAMS module.

order Specifies a decimal numeric value used to control the order by which the module's **`init`** and **`start`** routines are called, and the order of **`execsw`** entries. Higher-numbered values come first. For most modules, the order is unimportant, and this field should be 0.

bmaj Specifies the block major number(s) for this module. This field should contain either a single decimal number, or two numbers separated by a dash to indicate an inclusive range of values ("multiple majors"). The (first) value should normally be zero prior to installation with **`idinstall`**(1M). For example, to request four major numbers, this field would be initialized to "**`0-3`**" in the DSP **`Master`** file.

If the **`b`** flag is set—and the **`k`** flag is not set—in the *characteristics* field, **`idinstall`** will automatically assign block major numbers for the device. If the **`b`** flag and the **`k`** flags are both set, the *bmaj* field value will be used.

cmaj Specifies the character major number(s) for this module. This field should contain either a single decimal number, or two numbers separated by a dash to indicate an inclusive range of values ("multiple majors"). The (first) value should normally be zero prior to installation with **`idinstall`**(1M). For example, to request four major numbers, this field would be initialized to "**`0-3`**" in the DSP **`Master`** file.

If the **c** flag is set—and the **k** flag is not set—in the *characteristics* field, **idinstall** will automatically assign character major numbers for the device. If the **c** flag and the **k** flags are both set, the *cmaj* field value will be used.

NOTES

Specifying STREAMS Devices and Modules

STREAMS modules and drivers are treated in a slightly different way from other drivers, and their configuration reflects this difference. To specify a STREAMS device driver, its **Master** file should specify both an **S** and a **c** flag in the *characteristics* field. This indicates that it is a STREAMS driver and that it requires an entry in the *cdevsw* table, where STREAMS drivers are normally configured into the system.

A STREAMS module that is not a device driver, such as a line discipline module, requires both an **S** and an **m** flag in the *characteristics* field of its **Master** file; it should not specify a **c** flag, as a device driver does.

In cases where a module contains both a STREAMS module and a STREAMS driver, the **S**, **c** and **m** flags should all be specified.

Compatibility Considerations

For compatibility with existing add-on DSP packages, **idinstall** also accepts the old (version 0) **mdevice** file format, which had a single non-comment line that contained the following nine fields:

name funcs characteristics prefix bmaj cmaj min_unit max_unit dmachan

When presented with a version 0 **mdevice** file, **idinstall** converts the file to version 1 **Master** file format. During **mdevice** file conversion:

The *funcs* field is converted into **$entry** lines

The following *characteristics* flags are ignored as obsolete: **a, i, n, s, G, H, M, N, R** (the **f** flag in the version 0 **mdevice** file will still be recognized, but should not be used in a version 1 **Master** file)

The *min_unit* and *max_unit* fields are ignored as obsolete

The *dmachan* field is moved to the **System** file

Because cross-dependencies exist in the version 0 **mdevice** and **sdevice** files for exec modules, **idinstall** cannot convert these files to version 1 files. They must be converted manually before using **idinstall**.

Note that **idinstall** also accepts obsolete **mfsys** files and converts them to version 1 **Master** file format.

SEE ALSO

idbuild(1M), **idcheck**(1M), **idinstall**(1M), **Master**(4), **modadmin**(1M), **res_major**(4), **Space.c**(4), **System**(4)

menu(4)

NAME

menu - form description file for **menu**(1) command

DESCRIPTION

menu is a menu and form generator that creates file-driven, full-screen forms and menus for accepting user input and displaying information. The form or menu to be displayed is specified in a form description file that allows text, lists, input fields, contents of files, and output from commands to be displayed.

The form description file consists of a number of keywords, each denoting the start of a new section of the form description. Each section of the form description corresponds to a different part of the menu described. For example, there is a keyword **.top** which specifies that the text that follows will be placed at the top of the screen. The text in each section of the form description file can be hard-coded in the form description file, or it may be redirected from a file or command.

I/O Redirection

In the form description file, text may be included from another file. This is handled by specifying a less than (<) character in the first column of the form description file, followed by the name of a file to include. The text included from the file will be included verbatim; that is, no keywords will be processed, and no I/O redirection or command substitution will be performed on the input redirected from a file.

Command Substitution

In the form description file, text may be the output from a shell command. This is handled by enclosing a command to be executed in backquotes. Note that no more than one command will be parsed per input line of the form description file, and no command may span more than one line of the form description file. Again, the text included from the output of the command will be included verbatim; that is, no keywords will be processed, and no I/O redirection or command substitution will be performed on the input given by the output of a command.

Comments

The form description file may contain comments, which will be ignored. Comment lines are specified by placing a pound sign (#) in the first column.

SEE ALSO

menu(1), **menu_colors.sh**(1)

NAME

mkdev – file format for the **pdimkdev** utility

DESCRIPTION

The **pdimkdev** utility is executed when the system transitions from single-user to multi-user mode to create special device file entries for any newly configured PDI peripheral hardware. The **pdimkdev** utility allows for provision of special device file naming conventions by the application.

The naming convention for a PDI peripheral device is described in a **mkdev** template file. Additional **mkdev** template files may be provided for new device types that require a different naming convention than the one already being used.

Template File Overview

Each device type has a corresponding template file that is, by convention, placed in the directory **/etc/scsi/mkdev.d**. The template file allows for special device files to be created in up to four directories. These are the character and block special directories and the corresponding simple administration directories. The permissions of the special device files may also be specified.

Device Naming Template Syntax

The following tables show the syntax for the **mkdev** template files. The lines up to the separator "DATA" are for user messages that may be device specific.

STRING	MEANING
QUERY	If the QUERY string in the template file is a string other than "–", then the query will be issued to the user under low inode conditions. The set of device files to be created will be determined based on the response. The only currently known application for this feature is to determine whether a disk device is a boot device.
POSTMSG	The POSTMSG string is printed after the naming of a new device. For SCSI tape devices, this message is "–", which **mkdev** ignores. For disk devices, the message instructs the user to use sysadm partitioning if necessary.

The next line provides the directory names for the block and character devices as well as the simple administration equivalents for these. On the remaining lines the first three fields provide the key, minor number and mode. The fourth and fifth fields provide, respectively, the block and character device files. The sixth and seventh fields provide the block and character simple administration names, which are linked to the files in fields four and five, respectively. Below is a key showing the use and meaning of characters and keywords in the templates that follow.

KEY	The key field is a single character from the set "–", "M", or "Y". The meanings are explained below.
Y	Under low inode conditions (less than 200 available in the root filesystem) these device files will be created if answer to QUERY is "yes". For disks, this will denote Logical Unit (LU) is bootable and files must be created for a bootable LU.
M	Device file is mandatory and is created in low inode conditions.
–	Special device file is to be created under normal situations (that is, more than 200 available inodes on the root filesystem).

Minor Number Calculation

The information available to the **pdimkdev** utility about each device is host adaptor slot number (C), target controller number (T), Logical Unit number (L), and major number. A formula, using the information available to the **pdimkdev** utility, is provided in the template file and is used to generate the minor number. The major and minor number are used in calls to **mknod**(2) to create the special device file.

STRING	MEANING
MINOR	**:** **[***constant*| **C**| **T**| **L]** **[+**| **x**| ***]** **MINOR** The minor field is a string that may contain any of the characters 'C', 'T', or 'L'. When calculating the minor number, the values for these variables are substituted in the expression given in the string. The expression for the minor number is evaluated right to left. (For example, `3+L*16` is `19` when `L=1`)
C	Host adaptor slot number.
T	Target controller SCSI ID.
L	Logical Unit number.

REFERENCES

pdimkdev(1M)

NAME

mnttab – mounted file system table

SYNOPSIS

#include <sys/mnttab.h>

DESCRIPTION

The file **/etc/mnttab** contains information about devices that have been mounted by the **mount** command. The information is in the following structure, defined in **sys/mnttab.h**:

```
struct  mnttab {
        char    *mnt_special;
        char    *mnt_mountp;
        char    *mnt_fstype;
        char    *mnt_mntopts;
        char    *mnt_time;
};
```

The fields in the mount table are space-separated and show the block special device, the mount point, the file system type of the mounted file system, the mount options, and the time at which the file system was mounted.

NOTES

Do not store information in the **mnttab** file other than the fields described above; fields may be added to this file in future releases and are reserved for future use.

SEE ALSO

getmntent(3C), **mount**(1M), **setmnt**(1M)

Mtune(4)

NAME

Mtune – tunable parameter definitions

SYNOPSIS

Mtune

DESCRIPTION

One of the ID/TP kernel configuration files, an **Mtune** file contains definitions of tunable parameters, including default values, for a kernel module type. System-specific tunable values are stored in **stune**. When the **Mtune** component of a module's Driver Software Package (DSP) is installed, **idinstall**(1M) stores the module's **Mtune** file information in **/etc/conf/mtune.d/***module-name*, where the file *module-name* is the name of the module being installed. Package scripts should never access **/etc/conf/mtune.d** files directly; only the **idinstall** and **idtune** commands should be used.

Each tunable parameter is specified on a separate line of the form:

parameter-name default-value minimum-value maximum-value

All fields are positional and must be separated by white space. Blank lines and lines beginning with '**#**' or '*****' are considered comments and are ignored.

The **Mtune** file fields are:

parameter-name	A string (maximum of 20 characters) used to construct the preprocessor **#define**s that pass the value for this parameter to the system when the system is built [see **Space.c**(4)].
default-value	Specifies the default value for this tunable parameter. If the value is not overridden by the **stune**(4) file, this value will be used when the system is built.
minimum-value	Specifies the minimum allowable value for this tunable parameter. If the parameter is set in the **stune**(4) file, the configuration tools will check that the value specified in the **stune** file is equal to or greater than this value.
maximum-value	Specifies the maximum allowable value for this tunable parameter. If the parameter is set in the **stune**(4) file, the configuration tools will check that the value specified in the **stune** file is equal to or less than this value.

For detailed information on **Mtune** parameters, refer to the advanced features sections on tunable parameters in your system administration documentation.

NOTES

Compatibility Considerations

For compatibility with existing add-on DSP packages, **idbuild**(1M) maintains a flat file, **/etc/conf/cf.d/mtune**, which contains all of the tunable parameters. Packages may read this file to find existing values, and may add new tunables to the file. This mechanism is discouraged, however; new packages should use the **idinstall** and **idtune** commands.

SEE ALSO

idbuild(1M), **idinstall**(1M), **idtune**(1M), **Space.c**(4), **stune**(4)

NAME

netconfig – network configuration database

SYNOPSIS

#include <netconfig.h>

DESCRIPTION

The network configuration database, **/etc/netconfig**, is a system file used to store information about networks connected to the system and available for use. The **netconfig** database and the routines that access it [see **getnetconfig**(3N)] are part of the UNIX System V Network Selection component. The Network Selection component also includes the environment variable **NETPATH** and a group of routines that access the **netconfig** database using **NETPATH** components as links to the **netconfig** entries. **NETPATH** is described in **sh**(1); the **NETPATH** access routines are discussed in **getnetpath**(3N).

netconfig contains an entry for each network available on the system. Entries are separated by newlines. Fields are separated by whitespace and occur in the order in which they are described below. Whitespace can be embedded as "*blank*" or "*tab*." Backslashes may be embedded as "\\". Each field corresponds to an element in the **struct netconfig** structure. **struct netconfig** and the identifiers described on this manual page are defined in **/usr/include/netconfig.h**.

network ID

A string used to uniquely identify a network. *network ID* consists of non-null characters, and has a length of at least 1. No maximum length is specified. This namespace is locally significant and the local system administrator is the naming authority. All *network IDs* on a system must be unique.

semantics

The *semantics* field is a string identifying the "semantics" of the network, that is, the set of services it supports, by identifying the service interface it provides. The *semantics* field is mandatory. The following semantics are recognized.

tpi_clts	Transport Provider Interface, connectionless
tpi_cots	Transport Provider Interface, connection oriented
tpi_cots_ord	Transport Provider Interface, connection oriented, supports orderly release.
tpi_raw	Transport Provider Interface, raw

flag The *flag* field records certain two-valued ("true" and "false") attributes of networks. *flag* is a string composed of a combination of characters, each of which indicates the value of the corresponding attribute. If the character is present, the attribute is "true." If the character is absent, the attribute is "false." "-" indicates that none of the attributes is present. Two characters are currently recognized:

v Visible ("default") network. Used when the environment variable **NETPATH** is unset.

`b`	Enable RPC broadcast.

protocol family

The *protocol family* and *protocol name* fields are provided for protocol-specific applications.

The *protocol family* field contains a string that identifies a protocol family. The *protocol family* identifier follows the same rules as those for *network IDs*, that is, the string consists of non-null characters; it has a length of at least `1`; and there is no maximum length specified. A "-" in the *protocol family* field indicates that no protocol family identifier applies, that is, the network is experimental. The following are examples:

`loopback`	Loopback (local to host).
`inet`	Internetwork: UDP, TCP, and so on
`implink`	ARPANET imp addresses
`pup`	PUP protocols: for example, BSP
`chaos`	MIT CHAOS protocols
`ns`	XEROX NS protocols
`nbs`	NBS protocols
`ecma`	European Computer Manufacturers Association
`datakit`	DATAKIT protocols
`ccitt`	CCITT protocols, X.25, and so on
`sna`	IBM SNA
`decnet`	DECNET
`dli`	Direct data link interface
`lat`	LAT
`hylink`	NSC Hyperchannel
`appletalk`	Apple Talk
`nit`	Network Interface Tap
`ieee802`	IEEE 802.2; also ISO 8802
`osi`	Umbrella for all families used by OSI (for example, protosw lookup)
`x25`	CCITT X.25 in particular
`osinet`	AFI = 47, IDI = 4
`gosip`	U.S. Government OSI

protocol name

The *protocol name* field contains a string that identifies a protocol. The *protocol name* identifier follows the same rules as those for *network IDs*, that is, the string consists of non-NULL characters; it has a length of at least `1`; and there is no maximum length specified. The following protocol names are recognized. A "`-`" indicates that none of the names listed applies.

`tcp`	Transmission Control Protocol
`udp`	User Datagram Protocol
`icmp`	Internet Control Message Protocol

network device

The *network device* is the full pathname of the device used to connect to the transport provider. Typically, this device will be in the `/dev` directory. The *network device* must be specified.

directory lookup libraries

The *directory lookup libraries* support a "directory service" (a name-to-address mapping service) for the network. This service is implemented by the UNIX System V Name-to-Address Mapping feature. If a network is not provided with such a library, the *netdir* feature will not work. A "`-`" in this field indicates the absence of any lookup libraries, in which case name-to-address mapping for the network is non-functional. The directory lookup library field consists of a comma-separated list of full pathnames to dynamically linked libraries. Commas may be embedded as "`\,`"; backslashs as "`\\`".

Lines in **`/etc/netconfig`** that begin with a sharp sign (**#**) in column 1 are treated as comments.

The **`struct netconfig`** structure includes the following members corresponding to the fields in in the **`netconfig`** database entries:

`char * nc_netid`	Network ID, including NULL terminator
`unsigned long nc_semantics`	Semantics
`unsigned long nc_flag`	Flags
`char * nc_protofmly`	Protocol family
`char * nc_proto`	Protocol name
`char * nc_device`	Full pathname of the network device
`unsigned long nc_nlookups`	Number of directory lookup libraries
`char ** nc_lookups`	Full pathnames of the directory lookup libraries themselves
`unsigned long nc_unused[9]`	Reserved for future expansion (not advertised to user level)

The **`nc_semantics`** field takes the following values, corresponding to the semantics identified above:

```
NC_TPI_CLTS
NC_TPI_COTS
NC_TPI_COTS_ORD
NC_TPI_RAW
```

The **`nc_flag`** field is a bitfield. The following bit, corresponding to the attribute identified above, is currently recognized. **`NC_NOFLAG`** indicates the absence of any attributes.

```
NC_VISIBLE
```

FILES

`/etc/netconfig`
`/usr/include/netconfig.h`

SEE ALSO

`getnetconfig`(3N), **`getnetpath`**(3N), **`icmp`**(7), **`ip`**(7), **`netconfig`**(4), **`netdir_getbyname`**() [see **`netdir(3N)`**]

NAME

netdrivers – data file for networking boards to protocols mappings

SYNOPSIS

/etc/confnet.d/netdrivers

DESCRIPTION

The **netdrivers** file contains a list of hardware devices installed on the system and the protocols mapped to each board. The format of the file is:

<device><whitespace><protocol>

where *<device>* is the name of the device in **/dev** and *<protocol>* is a string matching the directory name in **/etc/confnet.d** where the **configure** script for the protocol can be found. Multiple lines will exist for any board that is being used for multiple protocols. Lines starting with "#" will be ignored.

Manipulation of the **netdrivers** file should be via the **netinfo** command.

Files

/etc/confnet.d/netdrivers

REFERENCES

generic **configure**(1M), INET-specific **configure**(1M), *protocol-specific* **configure**(1M), **interface**(4), **netdrivers**(4), **netinfo**(1M)

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 that the Class B network 128.32.0.0 should have eight bits of subnet field and eight bits of host field, in addition to the standard sixteen bits in the network field.

SEE ALSO

ifconfig(1M)

NAME

`netrc` – file for ftp remote login data

DESCRIPTION

The **`.netrc`** file contains data for logging in to a remote host over the network for file transfers by **`ftp`**(1). This file resides in the user's home directory on the machine initiating the file transfer. Its permissions should be set to disallow read access by group and others [see **`chmod`**(1)].

The following tokens are recognized; they may be separated by **`SPACE`**, **`TAB`**, or **`NEWLINE`** characters:

`machine` *name*
: Identify a remote machine name. The auto-login process searches the **`.netrc`** file for a **`machine`** token that matches the remote machine specified on the **`ftp`** command line or as an **`open`** command argument. Once a match is made, the subsequent **`.netrc`** tokens are processed, stopping when the EOF is reached or another **`machine`** token is encountered.

`login` *name*
: Identify a user on the remote machine. If this token is present, the auto-login process will initiate a login using the specified name.

`password` *string*
: Supply a password. If this token is present, the auto-login process will supply the specified string if the remote server requires a password as part of the login process. Note: if this token is present in the **`.netrc`** file, **`ftp`** will abort the auto-login process if the **`.netrc`** is readable by anyone besides the user.

`account` *string*
: Supply an additional account password. If this token is present, the auto-login process will supply the specified string if the remote server requires an additional account password, or the auto-login process will initiate an **`ACCT`** command if it does not.

`macdef` *name*
: Define a macro. This token functions 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

A **`.netrc`** file containing the following line:

```
machine ray login demo password mypassword
```

allows an autologin to the machine **`ray`** using the login name **`demo`** with password **`mypassword`**.

FILES

`~/.netrc`

SEE ALSO

`chmod`(1), `ftp`(1), `ftpd`(1M)

NAME

networks – network name data base

DESCRIPTION

The **networks** file contains information regarding the known networks which comprise the DARPA Internet. 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 SPACE and/or TAB characters. A '**#**' indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file. This file is normally created from the official network 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 (for example, **193.4.56.0**) using the **inet_network** routine from the Internet address manipulation library, **inet**(7). Network names may contain any printable character other than a field delimiter, NEWLINE, or comment character.

The **networks** file allows the usage of symbolic names instead of an IP address in the output of networking commands (for example, **netstat -r**).

FILES

/etc/networks

SEE ALSO

getnetent(3N), **inet**(7), **netstat**(1M)

NOTES

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

Node (4)

NAME

Node – device node definitions for a device driver

SYNOPSIS

Node

DESCRIPTION

One of the kernel configuration files, a **Node** file contains definitions used by the **idmknod**(1M) command to create the device nodes (block and character special files) associated with a device driver module. When the **Node** component of a module's Driver Software Package (DSP) is installed, **idinstall**(1M) stores the driver's **Node** file information in **/etc/conf/node.d/***module-name*, where *module-name* is the name of the driver being installed. Package scripts should never access **/etc/conf/node.d** files directly; the **idinstall** command should be used instead.

Each device node for the driver is specified on a separate line of the form:

module-name node-name type minor user group permissions

All fields are positional and must be separated by white space. The first four fields are required; the last three fields are optional. Blank lines and lines beginning with '**#**' or '*****' are considered comments and are ignored.

The **node** file fields are:

module-name Identifies the device to which this node applies. The name must match the name specified for the device in the *module-name* field of the **Master**(4) file. The device must be defined as a block and/or character device (**Master** file *characteristics* flag set to **b** or **c**). When the device node is created, the **Master** file *bmaj* or *cmaj* field values will be used as the major number for the created node.

node-name Specifies the name of the **Node** file to be created, relative to **/dev**. If this field specifies a pathname containing subdirectories, **idmknod** will automatically create these subdirectories.

type Identifies the type of node to be created. The character **b** indicates that the node is a block device; the character **c** indicates character device. The value for this field must match one of the flags specified for the device in the **mdevice** file *characteristics* field.

In cases where a device has multiple major numbers, the **Node** file *type* field must provide additional information used to identify which device nodes belong to which major. To do this, you specify a value of the form:

type:offset

where *type* is the type of node (**b** or **c**) and *offset* gives the offset to this particular device within the range of major numbers specified for this device type in the **Master** file. For example, the value "**c:2**" refers to a character major offset 2, which, given a major device type specification of "**15-18**", would translate to a character major number of 17.

minor Specifies the minor device number. This field can be coded in one of three ways:

> If this field specifies a (decimal) numeric value, the value is used as the minor device number for the created node.
>
> If this field specifies a non-numeric value, the value is assumed to be a request for a clone node, and the minor number will be set to the major number of the device whose *module-name* is the value of the field.
>
> If this field specifies a non-numeric value and the **Node** file *type* field specifies an offset value, the offset will be applied to the minor number instead of the major number when the node is created.

user This field is optional. If used, it specifies the user ID of the user that will own the node to be created. The user ID must be specified as decimal integer value.

group This field is optional. If used, it specifies the group ID of the group that will own the node to be created. The group ID must be specified as decimal integer value.

permissions This field is optional. If used, it specifies, in octal form, tye permissions for the node to be created, as given to the **chmod**(1) command (example: `0777`).

USAGE

Examples

The following sample **Node** file entries are provided as coding examples:

`iasy tty00 c 0`
Makes **`/dev/tty00`** for character device **`asy`** using minor device 0.

`clone net/nau/clone c nau`
Makes **`/dev/net/nau/clone`** for character device **`clone`**. The minor device number is set to the major device number of device **`nau`**.

`clone imx586_1 c:1 imx586`
Makes **`/dev/imx586_1`** for character device **`clone`**. The minor device number is set to the major device number of device **`imx586`** plus 1.

SEE ALSO

`idinstall`(1M), **`idmknod`**(1M)

NAME

`OlIcValues` – Input Context attribute names and value pairs

SYNOPSIS

```
#include <Xol/OpenLook.h>

typedef OlIcValues *OlIcValuesList;

typedef void  (*OlImProc)();
typedef void  *OlImValue;
```

DESCRIPTION

`OlIcValuesList` contains a list of Input Context attribute names and value pairs. It is used for getting and setting various Input Context attributes.

The `OlIcValues` structure includes the following members:

```
char *attr_name;
void *attr_value;
```

The `OlImCallback` structure includes the following members:

```
OlImValues  client_data;
OlImProc    callback;
```

The supported Input Context attributes are shown in the table below. The end of the list is indicated by a NULL value in the attribute name.

Input Context Attributes	
Attribute Name	Attribute Value Type
`OlNclientWindow`	`Window*`
`OlNClientArea`	`XRectangle*`
`OlNinputStyle`	`OlImStyle*`
`OlNfocusWindow`	`Window*`
`OlNpreeditArea`	`XRectangle*`
`OlNstatusArea`	`XRectangle*`
`OlNspotLocation`	`XPoint*`
`OlNresourceDatabse`	`XrmDatabase*`
`OlNpreeditAttributes`	`OlIcWindowAttr*`
`OlNstatusAttributes`	`OlIcWindowAttr*`

Attributes for Preedit and Status Windows	
Attribute Name	Attribute Value Type
`OlNcolormap`	`Colormap`
`OlNstdColormap`	`Colormap`
`OlNbackground`	`Pixel`
`OlNforeground`	`Pixel`
`OlNbackgroundPixmap`	`Pixmap`
`OlNfontset`	`OlFontList`
`OlNlineSpacing`	`int`
`OlNcursor`	`Cursor`
`Callbacks`	`OlImCallback`

`OlNclientWindow`
specifies the client window in which the Input Method may display data or create subwindows. Dynamic changes of client window are not supported; this argument must be set at the Input Context creation time and cannot be changed later. It is a static attribute required by **`OlCreateIc`**. The value is a pointer to a window.

`OlNClientArea`
specifies the client area in which the Input Method may display data or create subwindows. The Input Method will establish its own pre-edit and status geometry accordingly. When this attribute is left unspecified, The Input Method will default usable client area to actual client window geometry. This is a dynamic attribute that can be modified via calls to **`OlSetIcValues`**. The value is a pointer to an **`XRectangle`**.

`OlNinputStyle`
specifies the input style to be used. The value of this argument must be one of the supported styles returned by the **`OlGetImValues`** function, otherwise **`OlCreatIc`** will fail. If this attribute is unspecified, the Input Method uses an implementation defined default style. MooLIT does not support Dynamic changes of Input Method style. This argument must be set at the Input Context creation time and cannot be changed later. The value is a pointer to **`OlImStyle`**.

`OlNfocusWindow`
specifies to the Input Method the window XID of the focus window. The input method may affect that window: select events on it, send events to it, modify its properties, and grab the keyboard within that window.

When this attribute is unspecified, the Input Method will default from the focus window to the client window. Explicitly setting this attribute from a non NULL value to NULL, forces the Input Method to clear any displayed data in the status area corresponding to the focus window. This is a dynamic attribute that can be modified via calls to **`OlSetIcValues`**. The value is a pointer to a window.

`OlNpreeditArea`

the area where pre-edit data should be displayed. The value of this argument is a pointer to **`XRectangle`**, relative to the client window. The Input Method may create a preedit window in this area, using the specified geometry, as a child of a client window.

When **`OlNpreeditArea`** is unspecified, the Input Method will default from the preedit area to an implementation defined area. This area is contained within the client area.

If you specify this attribute for root or **`XimPreEditCallbacks`** Input Method, it is ignored.

If you specify this attribute for an **`XimPreEditArea`** Input Method, the width and height determine the size of the area within the "over-the-spot" window now available for pre-edit.

`OlNstatusArea`

specifies to the Input Method the usable area to display Input Context state information. The value of this argument is a pointer to **`XRectangle`**, relative to the client window.

The Input Method may or not create a status window in this area, using the specified geometry, as a child of the client window.

When **`OlNstatusArea`** is unspecified, the Input Method defaults to the status area defined by the Input Method implementation. The status area is contained within the client area. This is a dynamic attribute that can be modified via calls to **`OlSetIcValues`**.

Note that if a client leaves all areas unspecified, the Input Method may not be able to run properly. Some implementations will generate errors if none of the focus window, focus area, client area, preedit area, and status area are defined. At best, it may behave randomly using any area in the client window, possibly clearing the whole window or erasing any region.

`OlNspotLocation`

specifies the coordinates of the "spot" (the current cursor position in the text insertion window), to be used by the "over-the-spot" or "on-the-spot" Input Methods. The type is a pointer to **`Xpoint`**. The *x* coordinate specifies the position where the next character would be inserted. The *y* coordinate is the position of the baseline used by current text line in the focus window.

SEE ALSO

`OlCreateIc`(3Olit), **`OlDestroyIc`**(3Olit), **`OlGetIcValues`**(3Olit), **`OlImOfIc`**(3Olit), **`OlImValues`**(4), **`OlSetIcFocus`**(3Olit), **`OlSetIcValues`**(3Olit), **`OlUnsetIcFocus`**(3Olit), **`OlResetIc`**(3Olit)

NAME

OlImValues – a list of IM attributes

SYNOPSIS

#include <Xol/OpenLook.h>

DESCRIPTION

OlImValues contains a list of Input Method values or attributes returned by the **OlImStyles** structure. The **OlImStyles** structure includes the following members:

```
unsigned short count_styles; /* the number of input styles
supported */ OlImStyle *supported_styles;
```

count_styles is also the size of the array in the **supported_styles** field. Each element in the array represents a different input style supported by this Input Method. It is a bitmask in which the Input Method indicates its requirements, should this style be selected. These requirements fall into the following categories:

OlImPreEditArea	require the client to provide some area values for preediting. Refer to the Input Context attribute **OlNpreeditArea**.
OlIMPreditPosition	require the client to provide positional values. Refer to Input Context attributes **OlNspotLocation** and **OlN-focusWindow**.
OlImPreEditCallbacks	require the client to define the set of preedit callbacks. Refer to Input Context attributes **OlNPreEditStart-Callback**, **OlNPreEditDoneCallback**, **OlNPreEdit-DrawCallback**, and **OlNPreEditCaretCallback**.
OlImNeedNothing	function without any preedit values.
OlImStatusArea	require the client to provide some area values for status feedback. Refer to **OlNArea** and **OlNAreaNeeded**.
OlImStatusCallbacks	require the client to define the set of status callbacks.

SEE ALSO

OlCreateIc(3Olit), **OlDestroyIc**(3Olit), **OlGetIcValues**(3Olit), **OlIcValues**(4), **OlImOfIc**(3Olit), **OlSetIcFocus**(3Olit), **OlSetIcValues**(3Olit), **OlUnsetIcFocus**(3Olit), **OlResetIc**(3Olit)

NAME

`.ott` – FACE object architecture information

DESCRIPTION

The FACE object architecture stores information about object-types in an ASCII file named `.ott` (object type table) that is contained in each directory. This file describes all of the objects in that directory. Each line of the `.ott` file contains information about one object in pipe-separated fields. The fields are (in order):

name	the name of the actual UNIX System file.
dname	the name that should be displayed to the user, or a dot if it is the same as the name of the file.
description	the description of the object, or a dot if the description is the default (the same as object-type).
object-type	the FACE internal object type name.
flags	object specific flags.
mod time	the time that FACE last modified the object. The time is given as number of seconds since 1/1/1970, and is in hexadecimal notation.
object information	an optional field, contains a set of semi-colon separated *name=value* fields that can be used by FACE to store any other information necessary to describe this object.

FILES

`.ott` is created in any directory opened by FACE.

NAME

passwd – password file

SYNOPSIS

/etc/passwd

DESCRIPTION

/etc/passwd is an ASCII file that contains basic information about each user's account. This file contains a one-line entry, for each authorized user, of the form:

login_name : password : uid : gid : comment : home_dir : login_shell

where:

login_name	is the name specified by the user when logging in. This field contains no uppercase characters, should not be more than eight characters long, and should begin with a non-numeric character (that is, any alphabetic or special character except colon).
password	contains the character **x**. This field remains only for compatibility reasons. Password information is contained in the file **/etc/shadow** [see **shadow**(4)].
uid	is the user's numerical ID for the system, which should be unique.
gid	is the numerical ID of the group to which the user belongs.
comment	is any information you think might be useful to a user of this file which is not included elsewhere in the file.
home_dir	is the pathname of 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**.

Fields are separated by a colon; entries, by a new-line. Comment lines (lines preceded by the **#** (pound) character) are not allowed in the **/etc/passwd** file.

/etc/passwd has general read permission on all systems, and can be used by routines that map numerical user IDs to names.

FILES

/etc/passwd
/etc/shadow
/usr/lib/locale/*locale***/LC_MESSAGES/uxcore.abi**
language-specific message file [See **LANG** on **environ**(5).]

SEE ALSO

getpwent(3C), **group**(4), **login**(1), **passwd**(1), **putpwent**(3C), **pwconv**(1M), **shadow**(4), **unistd**(4), **useradd**(1M), **userdel**(1M), **usermod**(1M)

pathalias (4)

NAME

pathalias – alias file for FACE

DESCRIPTION

The **pathalias** files contain lines of the form **alias=***path* where *path* can be one or more colon-separated directories. Whenever a FACE user references a path not beginning with a "/," this file is checked. If the first component of the pathname matches the left-hand side of the equals sign, the right-hand side is searched much like **$PATH** variable in the UNIX System. This allows users to reference the folder **$HOME/FILECABINET** by typing **filecabinet**.

There is a system-wide **pathalias** file called **$VMSYS/pathalias**, and each user can also have local alias file called **$HOME/pref/pathalias**. Settings in the user alias file override settings in the system-wide file. The system-wide file is shipped with several standard FACE aliases, such as **filecabinet**, **wastebasket**, **preferences**, **other_users**, and so on.

NOTES

Unlike command keywords, partial matching of a path alias is not permitted, however, path aliases are case insensitive. The name of an alias should be alphabetic, and in no case can it contain special characters like "/," "\," or "=." There is no particular limit on the number of aliases allowed. Alias files are read once, at login, and are held in core until logout. Thus, if an alias file is modified during a session, the change will not take effect until the next session.

FILES

$HOME/pref/pathalias
$VMSYS/pathalias

NAME

pkginfo – package characteristics file

DESCRIPTION

pkginfo is an ASCII file that describes the characteristics of the package along with information that helps control the flow of installation. It is created by the software packag developer.

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

PARAM="*value*"

There is no required order in which the parameters must be specified within the file. Each parameter is described below. Only fields marked with an asterisk are mandatory.

PKG*
: **PKG** is the parameter to which you assign an abbreviation for the name of the package being installed. The abbreviation must be a short string (no more than nine characters long) and it must conform to file naming rules. All characters in the abbreviation must be alphanumeric and the first may not be numeric. **install**, **new**, and **all** are reserved abbreviations.

NAME*
: Text that specifies the package name (maximum length of 256 ASCII characters).

ARCH
: A comma-separated list of alphanumeric tokens that indicate the architecture (for example, **ARCH=m68k,i386**) associated with the package. The **pkgmk** tool may be used to create or modify this value when actually building the package. The maximum length of a token is 16 characters and it cannot include a comma. **ARCH** is not a mandatory field. Therefore, if it is not specified or if it is specified as **NULL**, it is ignored.

VERSION*
: Text that specifies the current version associated with the software package. The maximum length is 256 ASCII characters and the first character cannot be a left parenthesis. The **pkgmk** tool may be used to create or modify this value when actually building the package.

CATEGORY*
: A comma-separated list of categories under which a package may be displayed. There are six categories: "application", "graphics", "system", "utilities", "set", and "patch." If you choose, you can also assign a package to one or more categories that you define. Categories are case-insensitive and may contain only alphanumerics. Each category is limited in length to 16 characters.

: For a Set Installation Package (SIP), this field must have the value "set". A SIP is a special purpose package that controls the installation of a set of packages.

DESC	Text that describes the package (maximum length of 256 ASCII characters).
VENDOR	Used to identify the vendor that holds the software copyright (maximum length of 256 ASCII characters).
HOTLINE	Phone number and/or mailing address where further information may be received or bugs may be reported (maximum length of 256 ASCII characters).
EMAIL	An electronic address where further information is available or bugs may be reported (maximum length of 256 ASCII characters).
VSTOCK	The vendor stock number, if any, that identifies this product (maximum length of 256 ASCII characters).
CLASSES	A space-separated list of classes defined for a package. The order of the list determines the order in which the classes are installed. Classes listed first will be installed first (on a medium-by-medium basis). This parameter may be modified by the request script. In this way, the request script may be used to select which classes in the package get installed on the system.
ISTATES	A list of allowable run states for package installation (for example, **"S s 1"**).
RSTATES	A list of allowable run states for package removal (for example, **"S s 1"**).
BASEDIR	The pathname to a default directory where "relocatable" files may be installed. If **BASEDIR** is not specified and *basedir* in the **admin**(4) file (**/var/sadm/install/admin/default**) is set to **default**, then **BASEDIR** is set to **/** by default. An administrator can override the value of **BASEDIR** by setting *basedir* in the **admin** file.
ULIMIT	If set, this parameter is passed as an argument to the **ulimit** command, which establishes the maximum size of a file during installation.
ORDER	A list of classes defining the order in which they should be put on the medium. Used by **pkgmk** in creating the package. Classes not defined in this field are placed on the medium using the standard ordering procedures.
MAXINST	The maximum number of package instances that should be allowed on a machine at the same time. By default, only one instance of a package is allowed. This parameter must be set in order to have multiple instances of a package.
PSTAMP	Production stamp used to mark the **pkgmap** file on the output volumes. Provides a means for distinguishing between production copies of a version if more than one is in use at a time. If **PSTAMP** is not defined, the default is used. The default consists of the UNIX system machine name followed by the string **"***YYMMDDHHmm***"** (year, month, date, hour, minutes).

INTONLY	Indicates that the package should be installed interactively only when set to any non-NULL value.
PREDEPEND	Used to maintain compatibility with dependency checking on packages delivered earlier than System V Release 4. Pre-Release 4 dependency checks were based on whether or not the name file for the required package existed in the **/usr/options** directory. This directory is not maintained for Release 4 and later packages because the **depend** file is used for checking dependencies. However, entries can be created in this directory to maintain compatibility. This is done automatically by **pkgmk**. This field is to be assigned the package instance name of the package.
SERIALNUM	A serial number, if any, that uniquely identifies this copy of the package (maximum length of 256 ASCII characters).

EXAMPLES

Here is a sample **pkginfo** file:

```
PKG="oam"
NAME="OAM Installation Utilities"
VERSION="3"
VENDOR="AT&T"
HOTLINE="1-800-ATT-BUGS"
EMAIL="attunix!olsen"
VSTOCK="0122c3f5566"
CATEGORY="system.essential"
ISTATES="S 2"
RSTATES="S 2"
```

NOTES

Developers may define their own installation parameters by adding a definition to this file. A developer-defined parameter should begin with a capital letter.

SEE ALSO

admin(4), **pkgmk**(1)

NAME

pkgmap – package contents description file

DESCRIPTION

pkgmap is an ASCII file that provides a complete listing of the package contents. It is automatically generated by **pkgmk**(1) using the information in the **prototype** file.

Each entry in **pkgmap** describes a single "deliverable object file." A deliverable object file includes shell scripts, executable objects, data files, directories, and so on. The entry consists of several fields of information, each field separated by a space. The fields are described below and must appear in the order shown.

part A field designating the part number in which the object resides. A part is a collection of files, and is the atomic unit by which a package is processed. A developer can choose the criteria for grouping files into a part (for example, based on class). If no value is defined in this field, part 1 is assumed.

ftype A one-character field that indicates the file type. Valid values are:

`f`	a standard executable or data file
`e`	a file to be edited upon installation or removal
`v`	volatile file (one whose contents are expected to change)
`d`	directory
`x`	an exclusive directory (See NOTES)
`l`	linked file
`p`	named pipe
`c`	character special device
`b`	block special device
`i`	installation script or information file
`s`	symbolic link

Once a file has the file type attribute **v**, it will always be volatile. For example, if a file being installed already exists and has the file type attribute **v**, then even if the version of the file being installed is not specified as volatile, the file type attribute will remain volatile.

class The installation class to which the file belongs. This name must contain only alphanumeric characters and be no longer than 12 characters. It is not specified if the **ftype** is **i** (information file).

pathname The pathname where the object will reside on the target machine, such as **/usr/bin/mail**. Relative pathnames (those that do not begin with a slash) indicate that the file is relocatable.

For linked files (**ftype** is either **l** or **s**), pathname must be in the form of *path1*=*path2*, with *path1* specifying the destination of the link and *path2* specifying the source of the link.

For symbolically linked files, when *path2* is a relative pathname starting with **./** or **../**, *path2* is not considered relocatable. For example, if you enter a line such as

```
s /foo/bar/etc/mount=../usr/sbin/mount
```

path1 (**/foo/bar/etc/mount**) will be a symbolic link to **../usr/sbin/mount**.

pathname may contain variables which support relocation of the file. A *$parameter* may be embedded in the pathname structure. **$BASEDIR** can be used to identify the parent directories of the path hierarchy, making the entire package easily relocatable. Default values for *parameter* and **BASEDIR** must be supplied in the **pkginfo** file and may be overridden at installation.

Special characters, such as an equal sign (=), are included in pathnames by surrounding the entire pathname in single quotes (as in, for example, **'/usr/lib/~='**).

major	The major device number. The field is only specified for block or character special devices.
minor	The minor device number. The field is only specified for block or character special devices.
mode	The octal mode of the file (for example, 0664). A question mark (?) indicates that the mode will be left unchanged, implying that the file already exists on the target machine. This field is not used for linked files, packaging information files or non-installable files.
owner	The owner of the file (for example, **bin** or **root**). The field is limited to 14 characters in length. A question mark (?) indicates that the owner will be left unchanged, implying that the file already exists on the target machine. This field is not used for linked files or non-installable files. It is used optionally with a package information file. If used, it indicates with what owner an installation script will be executed. Can be a variable specification in the form of **$[A-Z]**. Will be resolved at installation time (see NOTES).
group	The group to which the file belongs (for example, "bin" or "sys"). The field is limited to 14 characters in length. A question mark (?) indicates that the group will be left unchanged, implying that the file already exists on the target machine. This field is not used for linked files or non-installable files. It is used optionally with a package information file. If used, it indicates with what group an installation script will be executed. Can be a variable assignment in the form of **$[A-Z]**. Will be resolved at installation time (see NOTES).
size	The actual size of the file in bytes. This field is not specified for named pipes, special devices, directories or linked files.
cksum	The checksum of the file contents. This field is not specified for named pipes, special devices, directories or linked files.
modtime	The time of last modification, as reported by the **stat**(2) function call. This field is not specified for named pipes, special devices, directories or linked files.

Each **pkgmap** must have one line that provides information about the number and maximum size (in 512-byte blocks) of parts that make up the package. This line is in the following format:

:*number_of_parts maximum_part_size*

Lines that begin with "**#**" are comment lines and are ignored.

When files are saved during installation before they are overwritten, they are normally just copied to a temporary pathname. However, for files whose mode includes execute permission (but which are not editable), the existing version is linked to a temporary pathname and the original file is removed. This allows processes which are executing during installation to be overwritten.

EXAMPLES

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

```
:2 500
1 i pkginfo 237 1179 541296672
1 b class1 /dev/diskette 17 134 0644 root other
1 c class1 /dev/rdiskette 17 134 0644 root other
1 d none bin 0755 root bin
1 f none bin/INSTALL 0755 root bin 11103 17954 541295535
1 f none bin/REMOVE 0755 root bin 3214 50237 541295541
1 l none bin/UNINSTALL=bin/REMOVE
1 f none bin/cmda 0755 root bin 3580 60325 541295567
1 f none bin/cmdb 0755 root bin 49107 51255 541438368
1 f class1 bin/cmdc 0755 root bin 45599 26048 541295599
1 f class1 bin/cmdd 0755 root bin 4648 8473 541461238
1 f none bin/cmde 0755 root bin 40501 1264 541295622
1 f class2 bin/cmdf 0755 root bin 2345 35889 541295574
1 f none bin/cmdg 0755 root bin 41185 47653 541461242
2 d class2 data 0755 root bin
2 p class1 data/apipe 0755 root other
2 d none log 0755 root bin 1 NULL NULL
2 v none log/logfile 0755 root bin 41815 47563 541461333
2 d none save 0755 root bin
2 d none spool 0755 root bin
2 d none tmp 0755 root bin
```

NOTES

The **pkgmap** file may contain only one entry per unique pathname.

An exclusive directory type (*file*) type **x**) specifies directories that are constrained to contain only files that appear in the installation software database (**/var/sadm/install/contents**). If there are other files in the directory, they will be removed by **pkgchk -fx** as described on the **pkgchk**(1M) manual page.

Variable specifications for the *owner* and *group* fields are defined in the **pkginfo** file. For example, *owner* could be **$OWNER** in the **pkgmap** file; if **OWNER** is defined as **root** in the **pkginfo** file, **$OWNER** will get the value **root** when the file is installed.

NAME

pnch – file format for card images

DESCRIPTION

The PNCH format is a convenient representation for files consisting of card images in an arbitrary code.

A PNCH file is a simple concatenation of card records. A card record consists of a single control byte followed by a variable number of data bytes. The control byte specifies the number (which must lie in the range 0–80) of data bytes that follow. The data bytes are 8-bit codes that constitute the card image. If there are fewer than 80 data bytes, it is understood that the remainder of the card image consists of trailing blanks.

NAME

priv – privilege data file

DESCRIPTION

A privilege data file entry has the following format:

dev **:** *fid* **:** *valid* **:** *%fixed privs* **:** *pathname*

Each field in the entry is separated by a colon (**:**) character. The *dev* (see **stat**(2)) and *fid* are, respectively, the ID of the device containing a directory entry for this file, and a unique *file* *id*entification number for this file. These are stored as decimal strings. The *valid* field is the last file status change time (see **stat**(2)) contained in the inode of the named file and stored as a decimal string. The *fixed* privileges field begins with a (%) delimiter and contains character strings which have comma (**,**) separated privilege names for each field. The *pathname* is the absolute path name of the privileged file.

The privilege data file is **/etc/security/tcb/privs**.

SEE ALSO

filepriv(1M), **initprivs**(1M), **intro**(2), **stat**(2)

NAME

PrivTable - privilege table

DESCRIPTION

PrivTable is a list of permissions that can be granted users through the desktop User_Setup client. Each entry includes a list of commands a user with permission will be able to execute with privilege, along with specific granted privileges. These privileges are granted the user using TFM (Trusted Facilities Management).

The privilege table contains lines of the format:

[*CatalogFile*:*Index*:]*CheckboxString*<TAB>*EntryList*<TAB>*HelpFile*

CheckboxString
: label to use for the checkbox in User_Setup. If the checkbox is checked, *EntryList* will be registered with TFM

[*CatalogFile*:*Index*:]
: translate *CheckboxString*. *CatalogFile* is the file of locale specific translations. *index* is the line number of the translation.

EntryList
: comma seperated list of entries. This takes the form

entryname1:*fullpath*:*priv1*:*priv2*:...,*entryname2* ...

HelpFile
: help file to use with this privilege checkbox. This may be a full path name or a locale-specific file in **/usr/X/lib/locale/***locale***/help/LoginMgr**.

Files

/etc/security/tfm/users/*user*
/usr/X/desktop/LoginMgr/PrivTable

SEE ALSO

dtprivilege(1M), **make-owner**(1M), **tfadmin**(1M)

NAME

proc – process file system

DESCRIPTION

/proc is a file system that provides access to the image of each active process in the system. The name of each entry in the **/proc** directory is a decimal number corresponding to the process ID. The owner of each "file" is determined by the process's user-ID.

Standard system call interfaces are used to access **/proc** files: **open**, **close**, **read**, **write**, and **ioctl**. An open for reading and writing enables process control; a read-only open allows inspection but not control. As with ordinary files, more than one process can open the same **/proc** file at the same time. Exclusive open is provided to allow controlling processes to avoid collisions: an **open** for writing that specifies **O_EXCL** fails if the file is already open for writing; if such an exclusive open succeeds, subsequent attempts to open the file for writing, with or without the **O_EXCL** flag, fail until the exclusively-opened file descriptor is closed. (Exception: a super-user open that does not specify **O_EXCL** succeeds even if the file is exclusively opened.) There can be any number of read-only opens, even when an exclusive write open is in effect on the file.

Data may be transferred from or to any locations in the traced process's address space by applying **lseek** to position the file at the virtual address of interest followed by **read** or **write**. The **PIOCMAP** operation can be applied to determine the accessible areas (mappings) of the address space. A contiguous area of the address space may appear as multiple mappings due to varying read/write/execute permissions. I/O transfers may span contiguous mappings. An I/O request extending into an unmapped area is truncated at the boundary.

Information and control operations are provided through **ioctl**. These have the form:

```
#include <sys/types.h>
#include <sys/signal.h>
#include <sys/fault.h>
#include <sys/syscall.h>
#include <sys/procfs.h>
void *p;
retval = ioctl(fildes, code, p);
```

The argument *p* is a generic pointer whose type depends on the specific **ioctl** code. Where not specifically mentioned below, its value should be zero. **sys/procfs.h** contains definitions of **ioctl** codes and data structures used by the operations. Certain operations can be performed only if the process file is open for writing; these include all operations that affect process control.

Process information and control operations involve the use of sets of flags. The set types **sigset_t**, **fltset_t**, and **sysset_t** correspond, respectively, to signal, fault, and system call enumerations defined in **sys/signal.h**, **sys/fault.h**, and **sys/syscall.h**. Each set type is large enough to hold flags for its own enumeration. Although they are of different sizes, they have a common structure and can be manipulated by these macros:

```
        prfillset(&set);            /* turn on all flags in set */
        premptyset(&set);           /* turn off all flags in set */
        praddset(&set, flag);       /* turn on the specified flag */
        prdelset(&set, flag);       /* turn off the specified flag */
        r = prismember(&set, flag); /* != 0 iff flag is turned on */
```

One of **prfillset** or **premptyset** must be used to initialize **set** before it is used in any other operation. **flag** must be a member of the enumeration corresponding to **set**.

The allowable **ioctl** codes follow. Those requiring write access are marked with an asterisk (*). Except where noted, an **ioctl** to a process that has terminated elicits the error **ENOENT**.

PIOCSTATUS

This returns status information for the process; *p* is a pointer to a **prstatus** structure:

```
typedef struct prstatus {
  long      pr_flags;      /* Process flags */
  short     pr_why;        /* Reason for process stop (if stopped) */
  short     pr_what;       /* More detailed reason */
  struct siginfo pr_info;  /* Info associated with signal or fault */
  short     pr_cursig;     /* Current signal */
  sigset_t  pr_sigpend;    /* Set of other pending signals */
  sigset_t  pr_sighold;    /* Set of held signals */
  struct sigaltstack pr_altstack; /* Alternate signal stack info */
  struct sigaction pr_action; /* Signal action for current signal */
  pid_t     pr_pid;        /* Process id */
  pid_t     pr_ppid;       /* Parent process id */
  pid_t     pr_pgrp;       /* Process group id */
  pid_t     pr_sid;        /* Session id */
  timestruc_t pr_utime;    /* Process user cpu time */
  timestruc_t pr_stime;    /* Process system cpu time */
  timestruc_t pr_cutime;   /* Sum of children's user times */
  timestruc_t pr_cstime;   /* Sum of children's system times */
  char      pr_clname[8];  /* Scheduling class name */
  long      pr_filler[20];/* Filler area for future expansion */
  long      pr_instr;      /* Current instruction */
  gregset_t pr_reg;        /* General registers */
} prstatus_t;
```

pr_flags is a bit-mask holding these flags:

PR_STOPPED	process is stopped
PR_ISTOP	process is stopped on an event of interest (see **PIOCSTOP**)
PR_DSTOP	process has a stop directive in effect (see **PIOCSTOP**)
PR_ASLEEP	process is in an interruptible sleep within a system call
PR_FORK	process has its inherit-on-**fork** flag set (see **PIOCSFORK**)
PR_RLC	process has its run-on-last-**close** flag set (see **PIOCSRLC**)

PR_PTRACE	process is being traced via **ptrace**
PR_PCINVAL	process program counter refers to an invalid address
PR_ISSYS	process is a system process (see **PIOCSTOP**)

pr_why and **pr_what** together describe, for a stopped process, the reason that the process is stopped. Possible values of **pr_why** are:

> **PR_REQUESTED** indicates that the process stopped because **PIOCSTOP** was applied; **pr_what** is unused in this case.
>
> **PR_SIGNALLED** indicates that the process stopped on receipt of a signal (see **PIOCSTRACE**); **pr_what** holds the signal number that caused the stop (for a newly-stopped process, the same value is in **pr_cursig**).
>
> **PR_FAULTED** indicates that the process stopped on incurring a hardware fault (see **PIOCSFAULT**); **pr_what** holds the fault number that caused the stop.
>
> **PR_SYSENTRY** and **PR_SYSEXIT** indicate a stop on entry to or exit from a system call (see **PIOCSENTRY** and **PIOCSEXIT**); **pr_what** holds the system call number.
>
> **PR_JOBCONTROL** indicates that the process stopped due to the default action of a job control stop signal (see **sigaction**); **pr_what** holds the stopping signal number.

pr_info, when the process is in a **PR_SIGNALLED** or **PR_FAULTED** stop, contains additional information pertinent to the particular signal or fault (see **sys/siginfo.h**).

pr_cursig names the current signal—that is, the next signal to be delivered to the process. **pr_sigpend** identifies any other pending signals. **pr_sighold** identifies those signals whose delivery is being delayed if sent to the process.

pr_altstack contains the alternate signal stack information for the process (see **sigaltstack**). **pr_action** contains the signal action information pertaining to the current signal (see **sigaction**); it is undefined if **pr_cursig** is zero.

pr_pid, **pr_ppid**, **pr_pgrp**, and **pr_sid** are, respectively, the process ID, the ID of the process's parent, the process's process group ID, and the process's session ID.

pr_utime, **pr_stime**, **pr_cutime**, and **pr_cstime** are, respectively, the user CPU and system CPU time consumed by the process, and the cumulative user CPU and system CPU time consumed by the process's children, in seconds and nanoseconds.

pr_clname contains the name of the process's scheduling class.

The **pr_filler** area is reserved for future use.

pr_instr contains the machine instruction to which the program counter refers. The amount of data retrieved from the process is machine-dependent. In general, the size is that of the machine's smallest instruction. If the program counter refers to an invalid address, **PR_PCINVAL** is set and **pr_instr** is undefined.

pr_reg is an array holding the contents of the general registers. The constants defined in **sys/regset.h** can be used as indices to refer to the general registers.

PIOCSTOP*, PIOCWSTOP

PIOCSTOP directs the process to stop and waits until it has stopped; **PIOCWSTOP** simply waits for the process to stop. These operations complete when the process stops on an event of interest, immediately if already so stopped. If *p* is non-zero it points to an instance of **prstatus_t** to be filled with status information for the stopped process.

An "event of interest" is either a **PR_REQUESTED** stop or a stop that has been specified in the process's tracing flags (set by **PIOCSTRACE**, **PIOCSFAULT**, **PIOCSENTRY**, and **PIOCSEXIT**). A **PR_JOBCONTROL** stop is specifically not an event of interest. (A process may stop twice due to a stop signal, first showing **PR_SIGNALLED** if the signal is traced and again showing **PR_JOBCONTROL** if the process is set running without clearing the signal.) If the process is controlled by **ptrace**, it comes to a **PR_SIGNALLED** stop on receipt of any signal; this is an event of interest only if the signal is in the traced signal set. If **PIOCSTOP** is applied to a process that is stopped, but not on an event of interest, the stop directive takes effect when the process is restarted by the competing mechanism; at that time the process enters a **PR_REQUESTED** stop before executing any user-level code.

ioctls are interruptible by signals so that, for example, an **alarm** can be set to avoid waiting forever for a process that may never stop on an event of interest. If **PIOCSTOP** is interrupted, the stop directive remains in effect even though the **ioctl** returns an error.

A system process (indicated by the **PR_ISSYS** flag) never executes at user level, has no user-level address space visible through **/proc**, and cannot be stopped. Applying **PIOCSTOP** or **PIOCWSTOP** to a system process elicits the error **EBUSY**.

PIOCRUN*

The traced process is made runnable again after a stop. If *p* is non-zero it points to a **prrun** structure describing additional actions to be performed:

```
typedef struct prrun {
  long      pr_flags;      /* Flags */
  sigset_t  pr_trace;      /* Set of signals to be traced */
  sigset_t  pr_sighold;    /* Set of signals to be held */
  fltset_t  pr_fault;      /* Set of faults to be traced */
  caddr_t   pr_vaddr;      /* Virtual address at which to resume */
  long      pr_filler[8];  /* Filler area for future expansion */
} prrun_t;
```

pr_flags is a bit-mask describing optional actions; the remainder of the entries are meaningful only if the appropriate bits are set in **pr_flags**. **pr_filler** is reserved for future use; this area must be filled with zeros by the user's program.

Flag definitions:

PRCSIG clears the current signal, if any (see **PIOCSSIG**).

PRCFAULT clears the current fault, if any (see **PIOCCFAULT**).

PRSTRACE sets the traced signal set to **pr_trace** (see **PIOCSTRACE**).

PRSHOLD sets the held signal set to **pr_sighold** (see **PIOCSHOLD**).

PRSFAULT sets the traced fault set to **pr_fault** (see **PIOCSFAULT**).

PRSVADDR sets the address at which execution resumes to **pr_vaddr**.

PRSTEP directs the process to single-step—i.e., to run and to execute a single machine instruction. On completion of the instruction, a hardware trace trap occurs. If **FLTTRACE** is being traced, the process stops, otherwise it is sent **SIGTRAP**; if **SIGTRAP** is being traced and not held, the process stops. This operation requires hardware support and may not be implemented on all processors.

PRSABORT is meaningful only if the process is in a **PR_SYSENTRY** stop or is marked **PR_ASLEEP**; it instructs the process to abort execution of the system call (see **PIOCSENTRY, PIOCSEXIT**).

PRSTOP directs the process to stop again as soon as possible after resuming execution (see **PIOCSTOP**). In particular if the process is stopped on **PR_SIGNALLED** or **PR_FAULTED**, the next stop will show **PR_REQUESTED**, no other stop will have intervened, and the process will not have executed any user-level code.

PIOCRUN fails (**EBUSY**) if applied to a process that is not stopped on an event of interest. Once **PIOCRUN** has been applied, the process is no longer stopped on an event of interest even if, due to a competing mechanism, it remains stopped.

PIOCSTRACE*
This defines a set of signals to be traced: the receipt of one of these signals causes the traced process to stop. The set of signals is defined via an instance of **sigset_t** addressed by *p*. Receipt of **SIGKILL** cannot be traced.

If a signal that is included in the held signal set is sent to the traced process, the signal is not received and does not cause a process stop until it is removed from the held signal set, either by the process itself or by setting the held signal set with **PIOCSHOLD** or the **PRSHOLD** option of **PIOCRUN**.

PIOCGTRACE
The current traced signal set is returned in an instance of **sigset_t** addressed by *p*.

PIOCSSIG*
The current signal and its associated signal information are set according to the contents of the **siginfo** structure addressed by *p* (see **sys/siginfo.h**). If the specified signal number is zero or if *p* is zero, the current signal is cleared. The semantics of this operation are different from those of **kill** or **PIOCKILL** in that the signal is delivered to the process immediately after execution is resumed (even if it is being held) and an additional **PR_SIGNALLED** stop does not intervene even if the signal is traced. Setting the current signal to **SIGKILL** terminates the process immediately, even if it is stopped.

PIOCKILL*
A signal is sent to the process with semantics identical to those of **kill**; *p* points to an **int** naming the signal. Sending **SIGKILL** terminates the process immediately.

PIOCUNKILL*

A signal is deleted, **i.e.** it is removed from the set of pending signals; the current signal (if any) is unaffected. *p* points to an **int** naming the signal. It is an error to attempt to delete **SIGKILL**.

PIOCGHOLD, PIOCSHOLD*

PIOCGHOLD returns the set of held signals (signals whose delivery will be delayed if sent to the process) in an instance of **sigset_t** addressed by *p*. **PIOCSHOLD** correspondingly sets the held signal set but does not allow **SIGKILL** or **SIGSTOP** to be held.

PIOCMAXSIG, PIOCACTION

These operations provide information about the signal actions associated with the traced process (see **sigaction**). **PIOCMAXSIG** returns, in the **int** addressed by *p*, the maximum signal number understood by the system. This can be used to allocate storage for use with the **PIOCACTION** operation, which returns the traced process's signal actions in an array of **sigaction** structures addressed by *p*. Signal numbers are displaced by 1 from array indices, so that the action for signal number *n* appears in position *n*–1 of the array.

PIOCSFAULT*

This defines a set of hardware faults to be traced: on incurring one of these faults the traced process stops. The set is defined via an instance of **fltset_t** addressed by *p*. Fault names are defined in **sys/fault.h** and include the following. Some of these may not occur on all processors; there may be processor-specific faults in addition to these.

FLTILL	illegal instruction
FLTPRIV	privileged instruction
FLTBPT	breakpoint trap
FLTTRACE	trace trap
FLTACCESS	memory access fault
FLTBOUNDS	memory bounds violation
FLTIOVF	integer overflow
FLTIZDIV	integer zero divide
FLTFPE	floating-point exception
FLTSTACK	unrecoverable stack fault
FLTPAGE	recoverable page fault

When not traced, a fault normally results in the posting of a signal to the process that incurred the fault. If the process stops on a fault, the signal is posted to the process when execution is resumed unless the fault is cleared by **PIOCCFAULT** or by the **PRCFAULT** option of **PIOCRUN**. **FLTPAGE** is an exception; no signal is posted. There may be additional processor-specific faults like this. **pr_info** in the **prstatus** structure identifies the signal to be sent and contains machine-specific information about the fault.

PIOCGFAULT

The current traced fault set is returned in an instance of **fltset_t** addressed by *p*.

PIOCCFAULT*
The current fault (if any) is cleared; the associated signal is not sent to the process.

PIOCSENTRY*, PIOCSEXIT*
These operations instruct the process to stop on entry to or exit from specified system calls. The set of syscalls to be traced is defined via an instance of **sysset_t** addressed by *p*.

When entry to a system call is being traced, the traced process stops after having begun the call to the system but before the system call arguments have been fetched from the process. When exit from a system call is being traced, the traced process stops on completion of the system call just prior to checking for signals and returning to user level. At this point all return values have been stored into the traced process's saved registers.

If the traced process is stopped on entry to a system call (**PR_SYSENTRY**) or when sleeping in an interruptible system call (**PR_ASLEEP** is set), it may be instructed to go directly to system call exit by specifying the **PRSABORT** flag in a **PIOCRUN** request. Unless exit from the system call is being traced the process returns to user level showing error **EINTR**.

PIOCGENTRY, PIOCGEXIT
These return the current traced system call entry or exit set in an instance of **sysset_t** addressed by *p*.

PIOCSFORK*, PIOCRFORK*
PIOCSFORK sets the inherit-on-fork flag in the traced process: the process's tracing flags are inherited by the child of a **fork**. **PIOCRFORK** turns this flag off: child processes start with all tracing flags cleared.

PIOCSRLC*, PIOCRRLC*
PIOCSRLC sets the run-on-last-close flag in the traced process: when the last writable **/proc** file descriptor referring to the traced process is closed, all of the process's tracing flags are cleared, any outstanding stop directive is canceled, and if the process is stopped, it is set running as though **PIOCRUN** had been applied to it. **PIOCRRLC** turns this flag off: the process's tracing flags are retained and the process is not set running when the process file is closed.

PIOCGREG, PIOCSREG*
These operations respectively get and set the saved process registers into or out of an array addressed by *p*; the array has type **gregset_t**. Register contents are accessible using a set of predefined indices (see **PIOCSTATUS**). For security, certain bits of the processor-status word cannot be modified by **PIOCSREG**. There may be other privileged registers that cannot be modified at all. **PIOCSREG** fails (**EBUSY**) if applied to a process that is not stopped on an event of interest.

PIOCGFPREG, PIOCSFPREG*
These operations respectively get and set the saved process floating-point registers into or out of a structure addressed by *p*; the structure has type **fpregset_t**. An error (**EINVAL**) is returned if there is no floating-point hardware on the machine. **PIOCSFPREG** fails (**EBUSY**) if applied to a process that is not stopped on an event of interest.

PIOCNICE*

The traced process's **nice** priority is incremented by the amount contained in the **int** addressed by *p*. Only the super-user may better a process's priority in this way, but any user may make the priority worse.

PIOCPSINFO

This returns miscellaneous process information such as that reported by **ps**(1). *p* is a pointer to a **prpsinfo** structure containing at least the following fields:

```
typedef struct prpsinfo {
  char    pr_state;  /* numeric process state (see pr_sname) */
  char    pr_sname;  /* printable character representing pr_state */
  char    pr_zomb;   /* !=0: process terminated but not waited for */
  char    pr_nice;   /* nice for cpu usage */
  u_long  pr_flag;   /* process flags */
  uid_t   pr_uid;    /* real user id */
  gid_t   pr_gid;    /* real group id */
  pid_t   pr_pid;    /* unique process id */
  pid_t   pr_ppid;   /* process id of parent */
  pid_t   pr_pgrp;   /* pid of process group leader */
  pid_t   pr_sid;    /* session id */
  caddr_t pr_addr;   /* physical address of process */
  long    pr_size;   /* size of process image in pages */
  long    pr_rssize; /* resident set size in pages */
  caddr_t pr_wchan;  /* wait addr for sleeping process */
  timestruc_t pr_start;  /* process start time, sec+nsec since epoch */
  timestruc_t pr_time;   /* usr+sys cpu time for this process */
  long    pr_pri;    /* priority, high value is high priority */
  char    pr_oldpri; /* pre-SVR4, low value is high priority */
  char    pr_cpu;    /* pre-SVR4, cpu usage for scheduling */
  dev_t   pr_ttydev; /* controlling tty device (PRNODEV if none) */
  char    pr_clname[8];  /* Scheduling class name */
  char    pr_fname[16];  /* last component of execed pathname */
  char    pr_psargs[PRARGSZ]; /* initial characters of arg list */
  long    pr_filler[20]; /* for future expansion */
} prpsinfo_t;
```

Some of the entries in **prpsinfo**, such as **pr_state** and **pr_flag**, are system-specific and should not be expected to retain their meanings across different versions of the operating system. **pr_addr** is a vestige of the past and has no real meaning in current systems.

PIOCPSINFO can be applied to a **zombie** process (one that has terminated but whose parent has not yet performed a **wait** on it).

PIOCNMAP, PIOCMAP

These operations provide information about the memory mappings (virtual address ranges) associated with the traced process. **PIOCNMAP** returns, in the **int** addressed by *p*, the number of mappings that are currently active. This can be used to allocate storage for use with the **PIOCMAP** operation, which returns the list of currently active mappings. For **PIOCMAP**, *p* addresses an array of elements of type **prmap_t**; one array element (one structure) is returned for each mapping, plus an additional element containing all zeros to mark the end of the list.

```
typedef struct prmap {
  caddr_t  pr_vaddr;       /* Virtual address base */
  u_long   pr_size;        /* Size of mapping in bytes */
  off_t    pr_off;         /* Offset into mapped object, if any */
  long     pr_mflags;      /* Protection and attribute flags */
  long     pr_filler[4];   /* Filler for future expansion */
} prmap_t;
```

pr_vaddr is the virtual address base (the lower limit) of the mapping within the traced process and **pr_size** is its size in bytes. **pr_off** is the offset within the mapped object (if any) to which the address base is mapped.

pr_mflags is a bit-mask of protection and attribute flags:

MA_READ	mapping is readable by the traced process
MA_WRITE	mapping is writable by the traced process
MA_EXEC	mapping is executable by the traced process
MA_SHARED	mapping changes are shared by the mapped object
MA_BREAK	mapping is grown by the **brk** system call
MA_STACK	mapping is grown automatically on stack faults

PIOCOPENM

The return value *retval* provides a read-only file descriptor for a mapped object associated with the traced process. If *p* is zero the traced process's **exec**ed file (its **a.out** file) is found. This enables a debugger to find the object file symbol table without having to know the path name of the executable file. If *p* is non-zero it points to a **caddr_t** containing a virtual address within the traced process and the mapped object, if any, associated with that address is found; this can be used to get a file descriptor for a shared library that is attached to the process. On error (invalid address or no mapped object for the designated address), –1 is returned.

PIOCCRED

Fetch the set of credentials associated with the process. *p* points to an instance of **prcred_t**, which is filled by the operation:

```
typedef struct prcred {
    uid_t    pr_euid;      /* Effective user id */
    uid_t    pr_ruid;      /* Real user id */
    uid_t    pr_suid;      /* Saved user id (from exec) */
    uid_t    pr_egid;      /* Effective group id */
    uid_t    pr_rgid;      /* Real group id */
    uid_t    pr_sgid;      /* Saved group id (from exec) */
    u_int    pr_ngroups;   /* Number of supplementary groups */
} prcred_t;
```

PIOCGROUPS

Fetch the set of supplementary group IDs associated with the process. *p* points to an array of elements of type **uid_t**, which will be filled by the operation. **PIOCCRED** can be applied beforehand to determine the number of groups (**pr_ngroups**) that will be returned and the amount of storage that should be allocated to hold them.

PIOCGETPR, PIOCGETU

These operations copy, respectively, the traced process's **proc** structure and user area into the buffer addressed by *p*. They are provided for completeness but it should be unnecessary to access either of these structures directly since relevant status information is available through other control operations. Their use is discouraged because a program making use of them is tied to a particular version of the operating system.

PIOCGETPR can be applied to a **zombie** process (see **PIOCPSINFO**).

NOTES

Each operation (**ioctl** or I/O) is guaranteed to be atomic with respect to the traced process, except when applied to a system process.

For security reasons, except for the super-user, an open of a **/proc** file fails unless both the user-ID and group-ID of the caller match those of the traced process and the process's object file is readable by the caller. Files corresponding to setuid and setgid processes can be opened only by the super-user. Even if held by the super-user, an open process file descriptor becomes invalid if the traced process performs an **exec** of a setuid/setgid object file or an object file that it cannot read. Any operation performed on an invalid file descriptor, except **close**, fails with **EAGAIN**. In this situation, if any tracing flags are set and the process file is open for writing, the process will have been directed to stop and its run-on-last-close flag will have been set (see **PIOCSRLC**). This enables a controlling process (if it has permission) to reopen the process file to get a new valid file descriptor, close the invalid file descriptor, and proceed. Just closing the invalid file descriptor causes the traced process to resume execution with no tracing flags set. Any process not currently open for writing via **/proc** but that has left-over tracing flags from a previous open and that **exec**s a setuid/setgid or unreadable object file will not be stopped but will have all its tracing flags cleared.

For reasons of symmetry and efficiency there are more control operations than strictly necessary.

FILES

/proc	directory (list of active processes)
/proc/*nnnnn*	process image

SEE ALSO

open(2), **ptrace**(2), **sigaction**(2), **signal**(2)

DIAGNOSTICS

Errors that can occur in addition to the errors normally associated with file system access:

ENOENT	The traced process has exited after being opened.
EIO	I/O was attempted at an illegal address in the traced process.
EBADF	An I/O or **ioctl** operation requiring write access was attempted on a file descriptor not open for writing.
EBUSY	**PIOCSTOP** or **PIOCWSTOP** was applied to a system process; an exclusive **open** was attempted on a process file already already open for writing; an **open** for writing was attempted and an exclusive open is in effect on the process file; **PIOCRUN**, **PIOCSREG** or **PIOCSFPREG** was

applied to a process not stopped on an event of interest; an attempt was made to mount **/proc** when it is already mounted.

EPERM	Someone other than the super-user attempted to better a process's priority by issuing **PIOCNICE**.
ENOSYS	An attempt was made to perform an unsupported operation (such as create, remove, link, or unlink) on an entry in **/proc**.
EFAULT	An I/O or `ioctl` request referred to an invalid address in the controlling process.
EINVAL	In general this means that some invalid argument was supplied to a system call. The list of conditions eliciting this error includes: the `ioctl` code is undefined; an `ioctl` operation was issued on a file descriptor referring to the **/proc** directory; an out-of-range signal number was specified with **PIOCSSIG**, **PIOCKILL**, or **PIOCUNKILL**; **SIGKILL** was specified with **PIOCUNKILL**; an illegal virtual address was specified in a **PIOCOPENM** request; **PIOCGFPREG** or **PIOCSFPREG** was issued on a machine without floating-point hardware.
EINTR	A signal was received by the controlling process while waiting for the traced process to stop via **PIOCSTOP** or **PIOCWSTOP**.
EAGAIN	The traced process has performed an **exec** of a setuid/setgid object file or of an object file that it cannot read; all further operations on the process file descriptor (except **close**) elicit this error.

NAME

profile – setting up an environment at login time

SYNOPSIS

/etc/profile
$HOME/.profile

DESCRIPTION

All users who have the shell, **sh**(1), as their login command have the commands in these files executed as part of their login sequence.

/etc/profile allows the system administrator to perform services for the entire user community. Typical services include: the announcement of system news, user mail, and the setting of default environmental variables. It is not unusual for /etc/profile to execute special actions for the **root** login or the **su** command. Computers running outside the U.S. Eastern time zone should have the line

```
. /etc/TIMEZONE
```

included early in /etc/profile [see timezone(4)].

The file $HOME/.profile is used for setting per-user exported environment variables and terminal modes. The following example is typical (except for the comments):

```
# Make some environment variables global
export MAIL PATH TERM
# Set file creation mask
umask 022
# Tell me when new mail comes in
MAIL=/var/mail/$LOGNAME
# Add my bin directory to the shell search sequence
PATH=$PATH:$HOME/bin
# Set terminal type
TERM=${L0:-u/n/k/n/o/w/n}
while :
do
     if [ -f ${TERMINFO:-/usr/share/lib/terminfo}/?/$TERM ]
     then break
     elif [ -f /usr/share/lib/terminfo/?/$TERM ]
     then break
     else echo "invalid term $TERM" 1>&2
     fi
     echo "terminal: \c"
     read TERM
done
# Set the erase character to backspace
stty erase '^H' echoe
```

FILES

`/etc/TIMEZONE`	timezone environment
`$HOME/.profile`	user-specific environment
`/etc/profile`	system-wide environment

SEE ALSO

env(1), **environ**(5), **login**(1), **mail**(1), **sh**(1), **stty**(1), **su**(1M), **term**(5), **terminfo**(4), **timezone**(4), **tput**(1)

NOTES

Care must be taken in providing system-wide services in **/etc/profile**. Personal **.profile** files are better for serving all but the most global needs.

NAME

protocols – protocol name data base

SYNOPSIS

/etc/protocols

DESCRIPTION

The **protocols** file contains information regarding the known protocols used in the DARPA Internet. 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.

The **protocols** file is used to initialize commands and protocols with these reserved values.

EXAMPLE

The following is a sample database:

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

FILES

/etc/protocols

SEE ALSO

getprotoent(3N)

NOTES

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

NAME

prototype – package information file

DESCRIPTION

prototype is an ASCII file used to specify package information. Each entry in the file describes a single deliverable object. An object may be a data file, directory, source file, executable object, etc. This file is generated by the package developer.

Entries in a **prototype** file consist of several fields of information separated by white space. Comment lines begin with a "**#**" and are ignored. The fields are described below and must appear in the order shown.

part An optional field designating the part number in which the object resides. A part is a collection of files, and is the atomic unit by which a package is processed. A developer can choose criteria for grouping files into a part (for example, based on class). If this field is not used, part 1 is assumed.

ftype A one-character field which indicates the file type. Valid values are:

f	a standard executable or data file
e	a file to be edited upon installation or removal
v	volatile file (one whose contents are expected to change)
d	directory
x	an exclusive directory (See NOTES)
l	linked file
p	named pipe
c	character special device
b	block special device
i	installation script or information file
s	symbolic link

Once a file has the file type attribute **v**, it will always be volatile. For example, if a file being installed already exists and has the file type attribute **v**, then even if the version of the file being installed is not specified as volatile, the file type attribute will remain volatile.

class The installation class to which the file belongs. This name must contain only alphanumeric characters and be no longer than 12 characters. The field is not specified for installation scripts. (**admin** and all classes beginning with capital letters are reserved class names.)

pathname The pathname where the file will reside on the target machine, for example, **/usr/bin/mail** or **bin/ras_proc**. Relative pathnames (those that do not begin with a slash) indicate that the file is relocatable. The form *path1*=*path2* may be used for two purposes: to define a link and to define local pathnames.

For linked files, *path1* indicates the destination of the link and *path2* indicates the source file. (This format is mandatory for linked files.)

For symbolically linked files, when *path2* is a relative pathname starting with **./** or **../**, *path2* is not considered relocatable. For example, if you enter a line such as

```
s /foo/bar/etc/mount=../usr/sbin/mount
```

path1 (**/foo/bar/etc/mount**) will be a symbolic link to **../usr/sbin/mount**.

For local pathnames, *path1* indicates the pathname an object should have on the machine where the entry is to be installed and *path2* indicates either a relative or fixed pathname to a file on the host machine which contains the actual contents.

A pathname may contain a variable specification, which will be resolved at the time of installation. This specification should have the form **$[A-Z]** (see NOTES).

Special characters, such as an equal sign (=), are included in pathnames by surrounding the entire pathname in single quotes (as in, for example, **'/usr/lib/~='**).

major
: The major device number. The field is only specified for block or character special devices.

minor
: The minor device number. The field is only specified for block or character special devices.

mode
: The octal mode of the file (for example, 0664). A question mark (**?**) indicates that the mode will be left unchanged, implying that the file already exists on the target machine. If the directory doesn't exist, the default is 0755. If it's a file, the default is 0644. This field is not used for linked files or packaging information files.

owner
: The owner of the file (for example, **bin** or **root**). The field is limited to 14 characters in length. A question mark (**?**) indicates that the owner will be left unchanged, implying that the file already exists on the target machine. If it doesn't exist, *owner* defaults to **root**. This field is not used for linked files or packaging information files.

  Can be a variable specification in the form of **$[A-Z]** (see NOTES). Will be resolved at installation time.

group
: The group to which the file belongs (for example, **bin** or **sys**). The field is limited to 14 characters in length. A question mark (**?**) indicates that the group will be left unchanged, implying that the file already exists on the target machine. If it doesn't exist, *group* defaults to **other**. This field is not used for linked files or packaging information files.

  Can be a variable specification in the form of **$[A-Z]** (see NOTES). Will be resolved at installation time.

An exclamation point (**!**) at the beginning of a line indicates that the line contains a command. These commands are used to incorporate files in other directories, to locate objects on a host machine, and to set permanent defaults. The following commands are available:

search
: Specifies a list of directories (separated by white space) to search for when looking for file contents on the host machine. The basename of the *path* field is appended to each directory in the ordered list until the file is located. This command should not be

specified in **prototype** files for packages that are to be compressed.

include Specifies a pathname which points to another prototype file to include. Note that **search** requests do not span **include** files.

default Specifies a list of attributes (mode, owner, group, mac, fixed, and inherited) to be used by default if attribute information is not provided for prototype entries which require the information. If either the mode, owner, or group attribute is supplied, all three of these attributes must be supplied.

The defaults do not apply to entries in **include** prototype files.

param=value Places the indicated parameter in the current environment.

The above commands may have variable substitutions embedded within them, as demonstrated in the two example **prototype** files below.

Before files are overwritten during installation, they are copied to a temporary pathname. The exception to this rule is files whose mode includes execute permission, unless the file is editable (that is, *ftype* is **e**). For files which meet this exception, the existing version is linked to a temporary pathname, and the original file is removed. This allows processes which are executing during installation to be overwritten.

EXAMPLES

Example 1:

```
!PROJDIR=/usr/proj
!BIN=$PROJDIR/bin
!CFG=$PROJDIR/cfg
!LIB=$PROJDIR/lib
!HDRS=$PROJDIR/hdrs
!search /usr/myname/usr/bin /usr/myname/src /usr/myname/hdrs
i pkginfo=/usr/myname/wrap/pkginfo
i depend=/usr/myname/wrap/depend
i version=/usr/myname/wrap/version
d none /usr/wrap 0755 root bin
d none /usr/wrap/bin 0755 root bin
! search $BIN
f none /usr/wrap/bin/INSTALL 0755 root bin
f none /usr/wrap/bin/REMOVE 0755 root bin
f none /usr/wrap/bin/addpkg 0755 root bin
!default 755 root bin
f none /usr/wrap/bin/audit
f none /usr/wrap/bin/listpkg
f none /usr/wrap/bin/pkgmk
# The logfile starts as a zero length file, since the source
# file has zero length. Later, the size of logfile grows.
v none /usr/wrap/logfile=/usr/wrap/log/zero_length 0644 root bin
# the following specifies a link (dest=src)
l none /usr/wrap/src/addpkg=/usr/wrap/bin/rmpkg
! search $SRC
```

```
!default 644 root other
f src /usr/wrap/src/INSTALL.sh
f src /usr/wrap/src/REMOVE.sh
f src /usr/wrap/src/addpkg.c
f src /usr/wrap/src/audit.c
f src /usr/wrap/src/listpkg.c
f src /usr/wrap/src/pkgmk.c
d none /usr/wrap/data 0755 root bin
d none /usr/wrap/save 0755 root bin
d none /usr/wrap/spool 0755 root bin
d none /usr/wrap/tmp 0755 root bin
d src /usr/wrap/src 0755 root bin
```

Example 2:

```
# this prototype is generated by 'pkgproto' to refer
# to all prototypes in my src directory
!PROJDIR=/usr/dew/projx
!include $PROJDIR/src/cmd/prototype
!include $PROJDIR/src/cmd/audmerg/protofile
!include $PROJDIR/src/lib/proto
```

SEE ALSO

pkginfo(4), **pkgmk**(1)

NOTES

Normally, if a file is defined in the **prototype** file but does not exist, that file is created at the time of package installation. However, if the file pathname includes a directory that does not exist, the file will not be created. For example, if the **prototype** file has the following entry:

```
f none /usr/dev/bin/command
```

and that file does not exist, it will be created if the directory **/usr/dev/bin** already exists or if the **prototype** also has an entry defining the directory:

```
d none /usr/dev/bin
```

An exclusive directory type (*file*) type **x**) specifies directories that are constrained to contain only files that appear in the software installation database (**/var/sadm/install/contents**). If there are other files in the directory, they will be removed by **pkgchk -fx** as described on the **pkgchk**(1M) man page.

Variable specifications for the **pathname**, *owner*, and *group* fields are defined in the **pkginfo** file. For example, *owner* could be **$OWNER** in the **pkgmap** file; if **OWNER** is defined as **root** in the **pkginfo** file, **$OWNER** will get the value **root** when the file gets installed.

publickey (4)

NAME

publickey – public key database

SYNOPSIS

`/etc/publickey`

DESCRIPTION

/etc/publickey is the public key database used for secure RPC. 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 a 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**(1M) command.

SEE ALSO

chkey(1), **newkey**(1M), **publickey**(3N)

NAME

`Rc` – system startup script

SYNOPSIS

`Rc`

DESCRIPTION

One of the kernel configuration files, an `Rc` file is an optional file that executes when the system is booted to initialize an installed kernel module. [Normally, this is a shell script (see **`sh`**(1).]

When the `Rc` component of a module's Driver Software Package (DSP) is installed, **`idinstall`**(1M) stores the module's `Rc` file in `/etc/conf/rc.d/`*module-name,* where *module-name* is the name of the module being installed. Package scripts should never access `/etc/conf/rc.d` files directly; the **`idinstall`** command should be used instead.

The contents of the `/etc/conf/rc.d` directory are linked to `/etc/idrc.d` whenever a new configuration of the kernel is first booted. On this initial reboot, and on all subsequent reboots, the module's `Rc` file is invoked upon entering **`init`** level 2 [see **`init`**(1M)].

SEE ALSO

`idinstall`(1M), **`init`**(1M), `Sd`(4)

res_major(4)

NAME

res_major – reserved major numbers for base system device drivers

SYNOPSIS

res_major

DESCRIPTION

One of the ID/TP kernel configuration files, the **res_major** file defines the major numbers that are reserved for use by device drivers supplied with the base system. When the **idinstall**(1M) command allocates major numbers for an add-on driver, it examines the contents of the file **/etc/conf/cf.d/res_major** to make sure the major number it intends to allocate to the add-on driver is not already reserved for allocation to a base system driver. This file should not be used by add-on drivers.

Any base system driver that sets a **k** (keep majors) flag in the *characteristics* field of its **Master**(4) file must add its major numbers to the **res_major** file. Each **res_major** file entry provides information about one type of base system driver, specified on a line of the form:

device-type major-number module-name

All fields are positional and must be separated by tabs.

The **res_major** file fields are:

device-type	Identifies the type of base system device driver. The character **b** indicates that the driver is a block device driver; the character **c** indicates that the driver is a character device driver. If the driver is both a block device driver and a character device driver, the driver must define two separate **res_major** entries, with one entry for each device type.
major-number	Specifies the major number(s) reserved by the base system device driver. If the device has multiple major numbers, this field should be specified as two numbers separated by a dash to indicate an inclusive range of reserved values. The value for this field must match the value specified in the *bmaj* or *cmaj* field in the driver's **Master** file.
module-name	Identifies the base system device driver for which the major number(s) are reserved. The name must match the name specified for the driver in the *module-name* field of the **Master** file.

SEE ALSO

idinstall(1M), **Master**(4)

NAME

`resolv.conf` – configuration file for name server routines

DESCRIPTION

The resolver configuration file contains information that is read by the resolver routines the first time they are invoked in a process. The file is designed to be human readable and contains a list of keyword-value pairs that provide various types of resolver information.

keyword   *value*

The different configuration options are:

`nameserver` *address*	The Internet address (in dot notation) of a name server that the resolver should query. At least one name server should be listed. Up to **`MAXNS`** (currently 3) name servers may be listed, in that case the resolver library queries tries them in the order listed. The algorithm used is to try a name server, and if the query times out, try the next, until out of name servers, then repeat trying all the name servers until a maximum number of retries are made.
`domain` *name*	The default domain to append to names that do not have a dot in them.
`address` *address*	An Internet address (in dot notation) of any preferred networks. The list of addresses returned by the resolver will be sorted to put any addresses on this network before any others.

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

`gethostent`(3N), **`named`**(1M), **`resolver`**(3N)

NAME

rfmaster – Remote File Sharing name server master file

DESCRIPTION

Each transport provider used by Remote File Sharing has an associated **rfmaster** file that identifies the primary and secondary name servers for that transport provider. The **rfmaster** file ASCII contains a series of records, each terminated by a newline; a record may be extended over more than one line by escaping the newline character with a backslash ("\"). The fields in each record are separated by one or more tabs or spaces. Each record has three fields:

name type data

The *type* field, which defines the meaning of the *name* and *data* fields, has three possible values. These values can appear in upper case or lower case:

p The **p** type defines the primary domain name server. For this type, *name* is the domain name and *data* is the full host name of the machine that is the primary name server. The full host name is specified as *domain.nodename.* There can be only one primary name server per domain.

s The **s** type defines a secondary name server for a domain. *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 The **a** type defines a network address for a machine. *name* is the full domain name for the machine and *data* is the network address of the machine. The network address can be in plain ASCII text or it can be preceded by a **\x** or **\X** to be interpreted as hexadecimal notation. (See the documentation for the particular network you are using to determine the network addresses you need.)

If a line in the **rfmaster** file begins with a **#** character, the entire line is treated as a comment.

There are at least two lines in the **rfmaster** file per domain name server: one **p** and one **a** line, to define the primary and its network address.

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 the associated **rfmaster** for a transport provider is missing, use

rfstart -p *primary_name_server_address*

to identify the primary name server's address for that transport provider. 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. There is an **rfmaster** file for each transport provider.

Files

/etc/rfs/*<transport>***/rfmaster**

USAGE

Examples

An example of an **rfmaster** file, using TCP/IP addresses, is shown below. (The network address shown are for illustration purposes only. Do not use these in your **rfmaster** file.)

```
ccs         p     ccs.comp1
ccs         s     ccs.comp2
ccs.comp2   a     \x00020ACEAE026E380000000000000000
ccs.comp1   a     \x00020ACEAE026E480000000000000000
```

REFERENCES

rfstart(1M)

NAME

routing – system supporting for packet network routing

DESCRIPTION

The network facilities provide general packet routing. Routing table maintenance may be implemented in applications processes.

A simple set of data structures compose a routing table used in selecting the appropriate network interface when transmitting packets. This table contains a single entry for each route to a specific network or host. The routing table was designed to support routing for the Internet Protocol (IP), but its implementation is protocol independent and thus it may serve other protocols as well. User programs may manipulate this data base with the aid of two **ioctl**(2) commands, **SIOCADDRT** and **SIOCDELRT**. These commands allow the addition and deletion of a single routing table entry, respectively. Routing table manipulations may only be carried out by privileged user.

A routing table entry has the following form, as defined in **/usr/include/net/route.h**:

```
struct rtentry {
    u_long  rt_hash;                    /* to speed lookups */
    struct  sockaddr rt_dst;            /* key */
    struct  sockaddr rt_gateway;        /* value */
    short   rt_flags;                   /* up/down?, host/net */
    short   rt_refcnt;                  /* # held references */
    u_long  rt_use;                     /* raw # packets forwarded */
#ifdef STRNET
    struct  ip_provider *rt_prov;       /* the answer: provider to use */
#else
    struct  ifnet *rt_ifp;              /* the answer: interface to use */
#endif /* STRNET */
};
```

with *rt_flags* defined from:

```
#define  RTF_UP       0x1           /* route usable */
#define  RTF_GATEWAY  0x2           /* destination is a gateway */
#define  RTF_HOST     0x4           /* host entry (net otherwise) */
```

Routing table entries come in three flavors: for a specific host, for all hosts on a specific network, for any destination not matched by entries of the first two types (a wildcard route). Each network interface installs a routing table entry when it it is initialized. Normally the interface specifies the route through it is a direct connection to the destination host or network. If the route is direct, the transport layer of a protocol family usually requests the packet be sent to the same host specified in the packet. Otherwise, the interface may be requested to address the packet to an entity different from the eventual recipient (that is, the packet is forwarded).

Routing table entries installed by a user process may not specify the hash, reference count, use, or interface fields; these are filled in by the routing routines. If a route is in use when it is deleted (**rt_refcnt** is non-zero), the resources associated with it will not be reclaimed until all references to it are removed.

User processes read the routing tables through the `/dev/kmem` device.

The *rt_use* field contains the number of packets sent along the route. This value is used to select among multiple routes to the same destination. When multiple routes to the same destination exist, the least used route is selected.

A wildcard routing entry is specified with a zero destination address value. Wildcard routes are used only when the system fails to find a route to the destination host and network. The combination of wildcard routes and routing redirects can provide an economical mechanism for routing traffic.

FILES

`/dev/kmem`

SEE ALSO

ioctl(2), **route**(1M), **routed**(1M)

DIAGNOSTICS

`EEXIST`	A request was made to duplicate an existing entry.
`ESRCH`	A request was made to delete a non-existent entry.
`ENOBUFS`	Insufficient resources were available to install a new route.

NAME

rpc – rpc program number data base

SYNOPSIS

rpc

DESCRIPTION

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

name of server for the RPC program
RPC program number
aliases

Items are separated by any number of blanks and/or tab characters. A # indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Below is an example of an RPC database:

```
#
#              rpc
#

rpcbind        100000        portmap sunrpc portmapper
rusersd        100002        rusers
nfs            100003        nfsprog
mountd         100005        mount showmount
walld          100008        rwall shutdown
sprayd         100012        spray
llockmgr       100020
nlockmgr       100021
status         100024
bootparam      100026
keyserv        100029        keyserver
```

NAME

rt_dptbl – real-time dispatcher parameter table

DESCRIPTION

The process scheduler (or dispatcher) is the portion of the kernel that controls allocation of the CPU to processes. The scheduler supports the notion of scheduling classes where each class defines a scheduling policy, used to schedule processes within that class. Associated with each scheduling class is a set of priority queues on which ready to run processes are linked. These priority queues are mapped by the system configuration into a set of global scheduling priorities which are available to processes within the class. (The dispatcher always selects for execution the process with the highest global scheduling priority in the system.) The priority queues associated with a given class are viewed by that class as a contiguous set of priority levels numbered from 0 (lowest priority) to *n* (highest priority—a configuration dependent value). The set of global scheduling priorities that the queues for a given class are mapped into might not start at zero and might not be contiguous (depending on the configuration).

The real-time class maintains an in-core table, with an entry for each priority level, giving the properties of that level. This table is called the real-time dispatcher parameter table (**rt_dptbl**). The **rt_dptbl** consists of an array of parameter structures (**struct rt_dpent**), one for each of the *n* priority levels. The properties of a given priority level *i* are specified by the *i*th parameter structure in this array (**rt_dptbl***i*).

A parameter structure consists of the following members. These are also described in the **/usr/include/sys/rt.h** header file.

rt_globpri	The global scheduling priority associated with this priority level. The mapping between real-time priority levels and global scheduling priorities is determined at boot time by the system configuration. The **rt_globpri** values cannot be changed with **dispadmin**(1M).
rt_quantum	The length of the time quantum allocated to processes at this level in ticks (HZ). The time quantum value is only a default or starting value for processes at a particular level as the time quantum of a real-time process can be changed by the user with the **priocntl** command or the **priocntl** system call.

An administrator can affect the behavior of the real-time portion of the scheduler by reconfiguring the **rt_dptbl**. There are two methods available for doing this.

DISPADMIN CONFIGURATION FILE

The **rt_quantum** values in the **rt_dptbl** can be examined and modified on a running system using the **dispadmin**(1M) command. Invoking **dispadmin** for the real-time class allows the administrator to retrieve the current **rt_dptbl** configuration from the kernel's in-core table, or overwrite the in-core table with values from a configuration file. The configuration file used for input to **dispadmin** must conform to the specific format described below.

Blank lines are ignored and any part of a line to the right of a # symbol is treated as a comment. The first non-blank, non-comment line must indicate the resolution to be used for interpreting the time quantum values. The resolution is specified as

RES=*res*

where *res* is a positive integer between 1 and 1,000,000,000 inclusive and the resolution used is the reciprocal of *res* in seconds. (For example, **RES=1000** specifies millisecond resolution.) Although very fine (nanosecond) resolution may be specified, the time quantum lengths are rounded up to the next integral multiple of the system clock's resolution. The system clock's resolution is hardware-dependent; this resolution can be calculated from the value of **HZ**, which is defined in the file **/usr/include/sys/param.h**. **HZ** gives the number of clock ticks per second of the system clock. For example, an **HZ** of 100 specifies 100 clock ticks per second, or one tick every 10 milliseconds (that is, this system clock has a resolution of 10 milliseconds). If the **-t** and **-r** options are used to specify a time quantum of 34 milliseconds, it is rounded up to 4 ticks (40 milliseconds) on a machine with an **HZ** of 100.

The remaining lines in the file are used to specify the **rt_quantum** values for each of the real-time priority levels. The first line specifies the quantum for real-time level 0, the second line specifies the quantum for real-time level 1, etc. There must be exactly one line for each configured real-time priority level. Each **rt_quantum** entry must be either a positive integer specifying the desired time quantum (in the resolution given by *res*), or the symbol **RT_TQINF** indicating an infinite time quantum for that level.

EXAMPLE

The following excerpt from a **dispadmin** configuration file illustrates the format. Note that for each line specifying a time quantum there is a comment indicating the corresponding priority level. These level numbers indicate priority within the real-time class, and the mapping between these real-time priorities and the corresponding global scheduling priorities is determined by the configuration specified in the **rt** master file. The level numbers are strictly for the convenience of the administrator reading the file and, as with any comment, they are ignored by **dispadmin** on input. **dispadmin** assumes that the lines in the file are ordered by consecutive, increasing priority level (from 0 to the maximum configured real-time priority). The level numbers in the comments should normally agree with this ordering; if for some reason they don't, however, **dispadmin** is unaffected.

```
# Real-Time Dispatcher Configuration File
RES=1000

#     TIME QUANTUM                    PRIORITY
#     (rt_quantum)                      LEVEL
           100                    #       0
           100                    #       1
           100                    #       2
           100                    #       3
           100                    #       4
           100                    #       5
            90                    #       6
            90                    #       7
            .                     .       .
            .                     .       .
            .                     .       .
            10                    #      58
            10                    #      59
```

FILES

`/usr/include/sys/rt.h`

SEE ALSO

dispadmin(1M), **priocntl**(1), **priocntl**(2)

Sassign (4)

NAME

Sassign – configurable device variables

SYNOPSIS

Sassign

DESCRIPTION

One of the kernel configuration files, the **Sassign** file gives system administrators the ability to assign specific actual devices to logical device names used by the kernel. One example is **rootdev**, which is the device that contains the root file system. At present, the **Sassign** file only supports block devices.

If the system administrator wants to assign a different actual device to perform a function, the administrator remaps the logical device name for that function to specify another configured device in the **Sassign** file. Note that the kernel must be rebuilt and rebooted for the new assignment to take affect.

Each logical device name in the **Sassign** file is specified (in **/etc/conf/sassign.d**) on a separate line of the form:

device-variable-prefix device-module-name minor node-name

All fields are positional and must be separated by white space. Blank lines and lines beginning with '**#**' or '*****' are considered comments and are ignored.

The **Sassign** file fields are:

device-variable-prefix
: Specifies a prefix identifier to be used to construct the logical name by which the device is known. When the kernel is rebuilt, the suffix **dev** will be appended to this identifier to form the full logical device name.

  The logical device name will be used to create a global variable of type **dev_t** during the kernel rebuild process. Any module that needs to reference the logical device should include an **extern dev_t** declaration for the logical device name variable.

device-module-name
: Identifies the name of actual device that is to perform the function associated with this logical device name. The name must match the name defined for the device in the *module-name* field of its **Master**(4) file.

minor
: Specifies the minor device number which is to be assigned to this logical device name. The major number for the logical device is the major number defined for the device identified in the *device-module-name* field.

node-name
: This field is used for the **swap** device only. The field specifies the full pathname of the block special file for the **swap** device.

NOTES

To create a variable which is intended to always refer to the same device, define it as a variable in the device's **Space.c** file, using the **PRFX_BMAJOR_***X* symbol from **/etc/conf/cf.d/config.h**, instead of using an **Sassign** file.

SEE ALSO

`idbuild`(1M), `Master`(4), `Space.c`(4)

sccsfile (4)

NAME

sccsfile – format of SCCS file

DESCRIPTION

An SCCS (Source Code Control System) file is an ASCII file. It consists of six logical parts: the checksum, the delta table (contains information about each delta), user names (contains login names and/or numerical group IDs of users who may add deltas), flags (contains definitions of internal keywords), comments (contains arbitrary descriptive information about the file), and the body (contains the actual text lines intermixed with control lines).

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 graphically as **@**. Any line described below that is not depicted as beginning with the control character is prevented from beginning with the control character.

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

Each logical part of an SCCS file is described in detail below.

Checksum

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

> **@h***DDDDD*

The value of the checksum is the sum of all characters, except those of the first line. The **@h** provides a magic number of (octal) 064001, depending on byte order.

Delta Table

The delta table consists of a variable number of entries of one of the following forms:

```
@s DDDDD/DDDDD/DDDDD
@d <type> <SCCS ID> yr/mo/da hr:mi:se <pgmr> DDDDD DDDDD
@i DDDDD ...
@x DDDDD ...
@g DDDDD ...
@m <MR number>
 ...
@c <comments> ...
 ...
@e
```

The first line (**@s**) contains the number of lines inserted/deleted/unchanged, respectively. The second line (**@d**) contains the type of the delta (normal: **D** or removed: **R**), the SCCS ID of the delta, the date and time of creation of the delta, the login 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 **@i**, **@x**, and **@g** lines contain the serial numbers of deltas included, excluded, and ignored, respectively. These lines are optional.

The **@m** lines (optional) each contain one MR number associated with the delta; the **@c** lines contain comments associated with the delta. The **@e** line ends the delta table entry.

User Names

The list of login names and/or numerical group IDs of users who may add deltas to the file, separated by new-lines. The lines containing these login names and/or numerical group IDs are surrounded by the bracketing lines **@u** and **@U**. An empty list allows anyone to make a delta. Any line starting with a **!** prohibits the succeeding group or user from making deltas.

Flags

Keywords used internally. See **admin**(1) for more information on their use. Each flag line takes the form:

> @f *<flag>*   *<optional text>*

The following flags are defined:

> @f t *<type of program>*
> @f v *<program name>*
> @f i *<keyword string>*
> @f b
> @f m *<module name>*
> @f f *<floor>*
> @f c *<ceiling>*
> @f d *<default-sid>*
> @f n
> @f j
> @f l *<lock-releases>*
> @f q *<user defined>*
> @f z *<reserved for use in interfaces>*

The **t** flag defines the replacement for the **%Y%** identification keyword. The **v** flag controls prompting for MR numbers in addition to comments; if the optional text is present it defines an MR number validity checking program. The **i** flag controls the warning/error aspect of the "**No id keywords**" message. When the **i** flag is not present, this message is only a warning; when the **i** flag is present, this message causes a fatal error (the file will not be "gotten", or the delta will not be made). When the **b** flag is present the **-b** keyletter may be used on the **get** command to cause a branch in the delta tree. The **m** flag defines the first choice for the replacement text of the **%M%** identification keyword. The **f** flag defines the floor release; the release below which no deltas may be added. The **c** flag defines the ceiling release; the release above which no deltas may be added. The **d** flag defines the default SID to be used when none is specified on a **get** command. The **n** flag causes **delta** 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). The absence of the **n** flag causes skipped releases to be completely empty. The **j** flag causes **get** to allow concurrent edits of the same base SID. The **l** flag defines a *list* of releases that are locked against editing. The **q** flag defines the replacement for the **%Q%** identification keyword. The **z** flag is used in specialized interface programs.

Comments

Arbitrary text is surrounded by the bracketing lines **@t** and **@T**. The comments section typically will contain a description of the file's purpose.

Body

The body consists of text lines and control lines. Text lines do not begin with the control character, control lines do. There are three kinds of control lines: insert, delete, and end, represented by:

@I *DDDDD*
@D *DDDDD*
@E *DDDDD*

respectively. The digit string is the serial number corresponding to the delta for the control line.

SEE ALSO

admin(1), **delta**(1), **get**(1), **prs**(1)

NAME

`Sd` – kernel module system shutdown script

SYNOPSIS

`Sd`

DESCRIPTION

One of the kernel configuration files, a **`Sd`** file is an optional file that executes when the system is shut down to perform any cleanup required for an installed kernel module. Normally, this is a shell script [see **`sh`**(1)].

When the **`Sd`** component of a module's Driver Software Package (DSP) is installed, **`idinstall`**(1M) stores the driver's **`Sd`** file information in **`/etc/conf/sd.d/`***module-name*, where *module-name* is the name of the module being installed. Package scripts should never access **`/etc/conf/sd.d`** files directly; only the **`idinstall`** command should be used.

The contents of the **`/etc/conf/sd.d`** directory are linked to **`/etc/idsd.d`** whenever a new configuration of the kernel is first booted. On the next system shutdown—and all subsequent system shutdowns—the module's **`Sd`** file is executed upon entering **`init`** level 0, 5, or 6 [see **`init`**(1M)].

SEE ALSO

`idinstall`(1M), **`init`**(1M), **`shutdown`**(1M), **`Rc`**(4)

NAME

services – Internet services and aliases

DESCRIPTION

The **services** file contains an entry for each service available through the DARPA Internet. 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 SPACE and/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 which search the file.

Service names may contain any printable character other than a field delimiter, NEWLINE, or comment character.

The **services** file is used to initialize commands and protocols with these traditional and reserved values.

FILES

/etc/services

SEE ALSO

getservent(3N), **inetd.conf**(4), RFC 1060

NAME

setinfo – set characteristics file

DESCRIPTION

setinfo is an ASCII file that describes the characteristics of the set along with information that helps control the flow of installation. It is created by the software set developer and is included in the Set Installation Package (SIP). A SIP is a special purpose package that controls the installation and removal of a set of packages.

Each entry in the **setinfo** file is a line that consists of predefined fields. Each entry corresponds to a package belonging to the set and must contain the following <tab>-separated fields:

1. *Package Abbr*

   This field contains the abbreviated name of the package. The abbreviation must be a short string (no more than nine characters long) and must conform to the file naming rules. All characters in the abbreviation must be alphanumeric and the first character cannot be numeric. **install**, **new**, and **all** are reserved.

   This abbreviated name must be the same as the one used in **pkginfo**(4).

2. *Parts* This field specifies the number of parts this package consists of.

3. *Default* This field contains the character 'y' indicating that the package is to be installed as a default. Conversely, an 'n' indicates that the package will not be installed.

4. *Category*

   The category under which the package belongs. Release 4 defines four categories: "application," "graphics," "system" and "utilities." All packages must be assigned to one of these categories. If you choose, you can also assign a package to a category you defined. Categories are case-insensitive and may contain only alphanumerics. Each category is limited to 16 characters.

5. *Package Full-Name*

   Text that specifies the package name (maximum length of 256 ASCII characters). This field must be the same as **NAME** in **pkginfo**(4).

EXAMPLES

setinfo file for set **admin**:

```
#ident  "@(#)set:cmn/set/admin/setinfo      1.2"
#ident  "$Header: $"

# Format for the setinfo file.  Field separator is: <tab>
# pkg parts default     category   pkg full-name
# abbr          y/n

oam   4    y    application     OA&M
bkrs  1    y    system          Extended Backup and Restore
face  1    y    application     FACE
```

NOTES

The order of the packages listed in the **setinfo** file must reflect any package dependencies (if any) and must represent the order in which the packages will occur on the media (in the case of datastream). Any package for which there exists a dependency must be listed prior to the package(s) that depends on it.

SEE ALSO

pkginfo(4)

NAME

setsize - disk space requirements file

DESCRIPTION

This set information file defines disk space requirements for the target environment. It contains information about all of the packages in the set. This file describes the disk space taken up by installed files as well as extra space needed for dynamically created files, as described in each package's **space**(4) file.

The generic format of a line in this file is:

pathname blocks inodes

Definitions for each field are as follows:

pkg	The short, or abbreviated, name of a package in the set. This name describes which package of the set requires the amount of space described by the rest of the data on this line in the **setsize** file.
pathname	Names a directory in which there are objects that will be installed or that will require additional space. The name may be the mountpoint for a filesystem. Names that do not begin with a slash (/) indicate relocatable directories.
blocks	Defines the number of 512 byte disk blocks required for installation of the files and directory entries contained in the pathname. (Do not include filesystem-dependent disk usage).
inodes	Defines the number of inodes required for the installation of the files and directory entries contained in the pathname.

At installation time, the set installation calls **setsizecvt**(1), which reduces the **setsize** file for a set to a **space**(4) file containing entries for only the packages that are selected. It is this resulting **space**(4) file against which space checking for the set is performed.

EXAMPLE

```
# space required by packages in the Networking Set
inet:/usr/adm   46   2
nfs:/etc   197   17
```

SEE ALSO

setsizecvt(1), **space**(4)

NAME

shadow – shadow password file

DESCRIPTION

/etc/shadow is an access-restricted ASCII system file that contains an entry for each user on the system. The fields within each entry are separated by colons; each entry is separated from the next by a new-line. Unlike the **/etc/passwd** file, **/etc/shadow** does not have general read permission.

Here are the fields in **/etc/shadow**:

login_name	The name by which a user identifies himself or herself when logging in.
password	A 13-character encrypted password for the user, a *lock* string to indicate the login is not accessible, or no string to show that there is no password for the login.
lastchanged	The number of days between January 1, 1970, and the date that the password was last modified.
minimum	The minimum number of days required between password changes.
maximum	The maximum number of days the password is valid.
warn	The number of days before password expires that the user is warned.
inactive	The number of days of inactivity allowed for that user.
expire	An absolute date specifying when the login may no longer be used.
flag	A character identifying a password generator.

The encrypted password consists of 13 characters chosen from a 64-character alphabet (**.**, **/**, **0-9**, **A-Z**, **a-z**).

To update this file, use the **passwd**, **useradd**, **usermod**, or **userdel** command.

FILES

/etc/shadow

SEE ALSO

getspent(3C), **login**(1), **passwd**(1), **passwd**(4), **putspent**(3C), **useradd**(1M), **userdel**(1M), **usermod**(1M)

NAME

sharetab – shared file system table

DESCRIPTION

sharetab resides in directory **/etc/dfs** and contains a table of local resources shared by the **share** command.

Each line of the file consists of the following fields:

pathname resource fstype specific_options description

where

pathname	Indicates the pathname of the shared resource.
resource	Indicates the symbolic name by which remote systems can access the resource.
fstype	Indicates the file system type of the shared resource.
specific_options	Indicates file-system-type-specific options that were given to the **share** command when the resource was shared.
description	Is a description of the shared resource provided by the system administrator when the resource was shared.

SEE ALSO

share(1M)

space(4)

NAME

space – disk space requirement file

DESCRIPTION

space is an ASCII file that gives information about disk space requirements for the target environment. It defines space needed beyond that which is used by objects defined in the **prototype** file—for example, files which will be installed with the **installf** command. It should define the maximum amount of additional space which a package will require.

The generic format of a line in this file is:

pathname blocks inodes

Definitions for the fields are as follows:

pathname	Specifies a directory name which may or may not be the mount point for a filesystem. Names that do not begin with a slash (/) indicate relocatable directories.
blocks	Defines the number of disk blocks required for installation of the files and directory entries contained in the pathname (using a 512-byte block size).
inodes	Defines the number of inodes required for installation of the files and directory entries contained in the pathname.

EXAMPLE

```
# extra space required by config data which is
# dynamically loaded onto the system
data 500  1
```

SEE ALSO

installf(1M), **prototype**(4)

NAME

`Space.c` – configuration-dependent kernel module data structure initializations

SYNOPSIS

`Space.c`

DESCRIPTION

One of the kernel configuration files, the **Space.c** file contains storage allocations and initializations for data structures associated with a kernel module, when the size or initial value of the data structures depend on configurable parameters, such as the number of subdevices configured for a particular device or a tunable parameter. For example, the **Space.c** file gives a driver the ability to allocate storage only for the subdevices actually being configured, by referencing symbolic constants defined in the **config.h** file.

When the **Space.c** component of a module's Driver Software Package (DSP) is installed, **idinstall**(1M) stores the module's **Space.c** file information in **/etc/conf/pack.d/***module-name***/space.c**, where *module-name* is the name of the module being installed.

Package scripts should never access **Space.c** files directly; only the **idinstall** command should be used.

The **config.h** file is a temporary file created in **/etc/conf/cf.d** during the system reconfiguration process. The file contains **#define** statements that can be used to specify the following information about the module:

```
Per module #defines:
---------------------------------------------------------------
#define  PRFX           Set to 1 if module is configured
#define  PRFX_CNTLS     Number of configured entries in System file
#define  PRFX_UNITS     Number of subdevices (sum of unit fields)
#define  PRFX_CMAJORS   Number of character major numbers supported
#define  PRFX_CMAJOR_0  Character major numbers supported; the first
                            major is PRFX_CMAJOR_0, the second
                            PRFX_CMAJOR_1, and so forth

Per instance #defines (PRFX_0 represents the first configured
instance, followed by PRFX_1, and so on if more than one
instance is configured):
----------------------------------------------------------------
#define  PRFX_0         Unit field value
#define  PRFX_0_VECT    Interrupt vector used
#define  PRFX_0_TYPE    Interrupt vector type
#define  PRFX_0_IPL     Interrupt priority level
#define  PRFX_0_SIOA    Starting input/output address
#define  PRFX_0_EIOA    Ending input/output address
#define  PRFX_0_SCMA    Starting controller memory address
#define  PRFX_0_ECMA    Ending controller memory address
#define  PRFX_0_CHAN    DMA channel used (-1 if none)
```

Since the module is installed as an object file, the module itself can not reference the **#defines** for the configurable device parameters in **config.h**. However, the module's **Space.c** is a C language source file, and as such, can define variables which can take on the values of the **#defines** in **config.h**. When the next system configuration is built, the **idbuild**(1M) command uses the list of arguments to **cc**(1) defined in **/etc/conf/cf.d/deflist** to compile the module's **Space.c** file before linking the module to the kernel.

NOTES

The following two **#defines** are generated for the **Space.c** file only if their values are identical for all instances:

```
#define  PRFX_CHAN     DMA channel used (-1 if none)
#define  PRFX_TYPE     Interrupt vector type used
```

SEE ALSO

idbuild(1M). **Master**(4), **System**(4)

EXAMPLES

```
#include <config.h>

        .
        .
        .

struct strtty lp_tty[LP_CNTLS];    /* tty structs for each device */
time_t  last_time[LP_CNTLS];       /* output char watchdog timeout */

        .
        .
        .

struct lpcfg lpcfg[LP_CNTLS] = {
        0,                          /* state */
        LP_0_SIOA+0,                /* data register port address */
        LP_0_SIOA+1,                /* status register port address */
        LP_0_SIOA+2,                /* control register port address */
        LP_0_VECT                   /* interrupt vector */
#ifdef LP_1
              ,
                                    /* next structure */
        0,
        LP_1_SIOA+0,
        LP_1_SIOA+1,
        LP_1_SIOA+2,
        LP_1_VECT
#endif     /* LP_1 */
#ifdef LP_2
              ,
                                /* next structure */
        0,
        LP_2_SIOA+0,
        LP_2_SIOA+1,
        LP_2_SIOA+2,
        LP_2_VECT
#endif      /* LP_2 */
        };
```

NAME

stat – (XENIX) data returned by **stat** system call

SYNOPSIS

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

DESCRIPTION

The system calls **stat, lstat** and **fstat** return data in a **stat** structure, which is defined in **stat.h**:

```
struct   stat
{
         dev_t     st_dev;
         ino_t     st_ino;
         mode_t    st_mode;
         nlink_t   st_nlink;
         uid_t     st_uid;
         gid_t     st_gid;
         dev_t     st_rdev;
         off_t     st_size;
         time_t    st_atime;
         time_t    st_mtime;
         time_t    st_ctime;
};
```

The constants used in the **st_mode** field are also defined in this file:

```
#define S_IFMT    0xF000    /* type of file */
#define S_IAMB    0x1FF     /* access mode bits */
#define S_IFIFO   0x1000    /* fifo */
#define S_IFCHR   0x2000    /* character special */
#define S_IFDIR   0x4000    /* directory */
#define S_IFNAM   0x5000    /* XENIX special named file */
#define S_INSEM   0x1       /* XENIX semaphore subtype of IFNAM */
#define S_INSEM   0x2       /* XENIX shared data subtype of IFNAM */
#define S_IFBLK   0x6000    /* block special */
#define S_IFREG   0x8000    /* regular */
#define S_IFLNK   0xA000    /* symbolic link */
#define S_ISUID   04000     /* set user id on execution */
#define S_ISGID   02000     /* set group id on execution */
#define S_ISVTX   01000     /* save swapped text even after use */
#define S_IREAD   00400     /* read permission, owner */
#define S_IWRITE  00200     /* write permission, owner */
#define S_IEXEC   00100     /* execute/search permission, owner */
#define S_ENFMT   S_ISGID   /* record locking enforcement flag */
#define S_IRWXU   00700     /* read, write, execute: owner */
#define S_IRUSR   00400     /* read permission: owner */
#define S_IWUSR   00200     /* write permission: owner */
#define S_IXUSR   00100     /* execute permission: owner */
```

```
#define S_IRWXG   00070    /* read, write, execute: group */
#define S_IRGRP   00040    /* read permission: group */
#define S_IWGRP   00020    /* write permission: group */
#define S_IXGRP   00010    /* execute permission: group */
#define S_IRWXO   00007    /* read, write, execute: other */
#define S_IROTH   00004    /* read permission: other */
#define S_IWOTH   00002    /* write permission: other */
#define S_IXOTH   00001    /* execute permission: other */
```

SEE ALSO

stat(2), **types**(5)

NAME

strcf – STREAMS Configuration File for STREAMS TCP/IP

DESCRIPTION

/etc/strcf contains the script that is executed by **slink**(1M) to perform the STREAMS configuration operations required for STREAMS TCP/IP.

The standard **/etc/strcf** file contains several functions that perform various configuration operations, along with a sample **boot** function. Normally, only the **boot** function must be modified to customize the configuration for a given installation. In some cases, however, it may be necessary to change existing functions or add new functions.

The following functions perform basic linking operations:

The **tp** function is used to set up the link between a transport provider, such as TCP, and IP.

```
#
# tp - configure transport provider (that is, tcp, udp, icmp)
# usage: tp devname
#
tp {
     p = open $1
     ip = open /dev/ip
     link p ip
}
```

The **linkint** function links the specified streams and does a **sifname** operation with the given name.

```
#
# linkint - link interface to ip or arp
# usage: linkint top bottom ifname
#
linkint {
     x = link $1 $2
     sifname $1 x $3
}
```

The **aplinkint** function performs the same function as **linkint** for an interface that uses the **app** module.

```
#
# aplinkint - like linkint, but app is pushed on dev
# usage: aplinkint top bottom ifname
#
aplinkint {
     push $2 app
     linkint $1 $2 $3
}
```

The following functions are used to configure different types of Ethernet interfaces:

The **uenet** function is used to configure an Ethernet interface for a cloning device driver that uses the *unit select* ioctl to select the desired interface. The interface name is constructed by concatenating the supplied prefix and the unit number.

```
#
# uenet - configure ethernet-type interface for cloning driver using
#         unit select
# usage: uenet ip-fd devname ifprefix unit
#
uenet {
      ifname = strcat $3 $4
      dev = open $2
      unitsel dev $4
      aplinkint $1 dev ifname
      dev = open $2
      unitsel dev $4
      arp = open /dev/arp
      linkint arp dev ifname
}
```

The **denet** function performs the same function as **uenet**, except that DL_ATTACH is used instead of *unit select*.

```
#
# denet - configure ethernet-type interface for cloning driver using
#         DL_ATTACH
# usage: denet ip-fd devname ifprefix unit
#
denet {
      ifname = strcat $3 $4
      dev = open $2
      dlattach dev $4
      aplinkint $1 dev ifname
      dev = open $2
      dlattach dev $4
      arp = open /dev/arp
      linkint arp dev ifname
}
```

The **cenet** function is used to configure an Ethernet interface for a cloning device driver that uses a different major number for each interface. The device name is formed by concatenating the supplied device name prefix and the unit number. The interface name is formed in a similar manner using the interface name prefix.

```
#
# cenet - configure ethernet-type interface for cloning driver with
#         one major per interface
# usage: cenet ip-fd devprefix ifprefix unit
#
cenet {
      devname = strcat $2 $4
```

```
          ifname = strcat $3 $4
          dev = open devname
          aplinkint $1 dev ifname
          dev = open devname
          arp = open /dev/arp
          linkint arp dev ifname
     }
```

The **senet** function is used to configure an Ethernet interface for a non-cloning device driver. Two different device nodes must be specified for IP and ARP.

```
     #
     # senet - configure ethernet-type interface for non-cloning driver
     # usage: senet ip-fd ipdevname arpdevname ifname
     #
     senet {
          dev = open $2
          aplinkint $1 dev $4
          dev = open $3
          arp = open /dev/arp
          linkint arp dev $4
     }
```

The **senetc** function is like **senet**, except that it allows the specification of a convergence module to be used with the Ethernet driver (such as, for the 3B2 emd driver).

```
     #
     # senetc - configure ethernet-type interface for non-cloning driver
     #          using convergence module
     # usage: senetc ip-fd convergence ipdevname arpdevname ifname
     #
     senetc {
          dev = open $3
          push dev $2
          aplinkint $1 dev $5
          dev = open $4
          push dev $2
          arp = open /dev/arp
          linkint arp dev $5
     }
```

The **loopback** function is used to configure the loopback interface.

```
     #
     # loopback - configure loopback device
     # usage: loopback ip-fd
     #
     loopback {
          dev = open /dev/loop
          linkint $1 dev lo0
     }
```

The **slip** function is used to configure a SLIP interface. This function is not normally executed at boot time.

```
#
# slip - configure slip interface
# usage: slip unit
#
slip {
     ip = open /dev/ip
     s = open /dev/slip
     ifname = strcat sl $1
     unitsel s $1
     linkint ip s ifname
}
```

The **boot** function is called by default when **slink** is executed. Normally, only the *interfaces* section and possibly the *queue params* section will have to be customized for a given installation. Examples are provided for the various Ethernet driver types.

```
#
# boot - boot time configuration
#
boot {
     #
     # queue params
     #
#    initqp /dev/udp rq 8192 40960
#    initqp /dev/ip muxrq 8192 40960 rq 8192 40960
     #
     # transport
     #
     tp /dev/tcp
     tp /dev/udp
     tp /dev/icmp
     tp /dev/rawip
}
```

FILES

/etc/strcf

SEE ALSO

slink(1M)

NAME

`strftime` – language-specific strings

DESCRIPTION

There can exist one printable file per locale to specify its date and time formatting information. These files must be kept in the directory `/usr/lib/locale/`*<locale>*`/LC_TIME`. The contents of these files are:

1. abbreviated month names (in order)
2. month names (in order)
3. abbreviated weekday names (in order)
4. weekday names (in order)
5. default strings that specify formats for locale time (`%X`) and locale date (`%x`)
6. default format for `cftime`, if the argument for `cftime` is zero or null
7. ante meridian string
8. post meridian string
9. default format for date command output

Each string is on a line by itself. All white space is significant. The order of the strings in the above list is the same order in which they must appear in the file.

EXAMPLE

Here are the contents of `/usr/lib/locale/C/LC_TIME`:

```
Jan
Feb
...
January
February
...
Sun
Mon
...
Sunday
Monday
...
%H:%M:%S
%m/%d/%y
%a %b %d %T %Z %Y
AM
PM
%a %b %d %T %Z %Y
```

FILES

`/usr/lib/locale/`*<locale>*`/LC_TIME`

SEE ALSO

`ctime`(3C), `setlocale`(3C), `strftime`(3C)

NAME

`Stubs.c` – stubs for kernel module symbols

SYNOPSIS

`Stubs.c`

DESCRIPTION

One of the kernel configuration files, a **`Stubs.c`** file is an optional C language source file that can be installed and compiled into the system as a "place holder" for a kernel module that will not be installed in the system. Its purpose is to enable the kernel to resolve references to the absent module's symbols.

When the **`Stubs.c`** component of a module's Driver Software Package (DSP) is installed, **`idinstall`**(1M) stores the module's **`Stubs.c`** file information in **`/etc/conf/pack.d/`***module-name***`/stubs.c`** where *module-name* is the name of the module being installed. Package scripts should never access **`Stubs.c`** files directly; only the **`idinstall`** command should be used.

A module's **`Stubs.c`** file contains function name and variable definition stubs for symbols defined in the module that can be referenced by other kernel modules being configured into the system. At compile time, the definitions in the **`Stubs.c`** file give the kernel the ability to resolve references made to the absent module's symbols.

SEE ALSO

`idinstall`(1M)

NAME

stune – local system settings for tunable parameters

SYNOPSIS

stune
stune.current

DESCRIPTION

The **/etc/conf/cf.d/stune** file contains tunable parameters for the kernel modules to be configured into the next system to be built [see **idbuild**(1M)]. The parameter settings in the **stune** file are used to override the default values specified in the **Mtune** file.

The contents of the **stune** file will only affect the next kernel rebuild. Once the new kernel has been installed to **/stand** and booted, the **stune** file is copied to **stune.current**. Any change made to the **stune.current** file using the **idtune**(1M) command with the **-c** option will affect all the loadable kernel modules subsequently configured into the running system.

Package scripts should never access **/etc/conf/cf.d/stune** or **/etc/conf/cf.d/stune.current** files directly; only the **idtune**(1M) command should be used.

The **stune** and **stune.current** files contain one line for each parameter to be set. Each line contains two positional fields separated by white space:

parameter-name new-value

Blank lines and lines beginning with '**#**' or '*****' are considered comments and are ignored.

The **stune** and **stune.current** fields are:

parameter-name	The name of the tunable parameter, as defined in the **Mtune** file.
new-value	Specifies the new value to be used to override the default value specified for this tunable parameter in the **Mtune** file. This value must fall within the valid range of values specified for this parameter in the **Mtune** file.

For detailed information on **stune** parameters, refer to the advanced features sections on tunable parameters in your system administration documentation.

SEE ALSO

idbuild(1M), **idtune**(1M), **Mtune**(4)

NAME

su – **su** options file

DESCRIPTION

Options for the su command [see **su**(1M)] can be set or changed with keywords in **/etc/default/su**. The following keywords are recognized by **su**:

SULOG=*filename*	Log (in *filename*) successful and unsuccessful attempts to execute **su**.
CONSOLE=*device*	If a user executes **su** to become a privileged user on a device other than *device*, a printed message will appear on *device* to inform the administrator of that fact.
PATH=*path_list*	When a user executes **su** to become an unprivileged user, the user's path will be set to *path_list*. The default is **/usr/bin:/usr/ccs/bin**.
SUPATH=*path_list*	When a user executes **su** to become a privileged user, the user's path will be set to *path_list*. The default is **/sbin:/usr/sbin:/usr/bin:/etc:/usr/ccs/bin**.
PROMPT:	If this parameter exists and is set to **No**, the **su** command does not prompt for a password (even if one is defined for *login_name*). The invoking user, however, must still have appropriate privilege to execute **su** successfully. If this parameter does not exist, or is set to anything other than **No** (including **NULL**), **su** prompts for a password when invoked and validates the password (if one is defined for *login_name*).

FILES

/etc/default/su

SEE ALSO

su(1M)

NAME

syslog.conf – (BSD) configuration file for syslogd system log daemon

SYNOPSIS

/etc/syslog.conf

DESCRIPTION

The file **/etc/syslog.conf** contains information used by the system log daemon, **syslogd**(1M), to forward a system message to appropriate log files and/or users. **syslog** preprocesses this file through **m4**(1) 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**(1M), **routed**(1M), and so on.
auth	The authorization system: **login**(1), **su**(1M), **getty**(1M), and so on.
lpr	The line printer spooling system: **lpr**(1), **lpc**(1M), and so on.
news	Reserved for the USENET network news system.
uucp	Reserved for the UUCP system; it does not currently use the **syslog** mechanism.
cron	The **cron/at** facility; **crontab**(1), **at**(1), **cron**(1M), and so on.
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 nonwhite character is a '#' are treated as comments.

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`

SEE ALSO

at(1), **cron**(1M), **crontab**(1), **getty**(1M), **login**(1), **lp**(1), **m4**(1), **syslog**(3), **syslogd**(1M), **su**(1M)

NAME

System – system-specific configuration information for a kernel module

SYNOPSIS

System

DESCRIPTION

One of the ID/TP kernel configuration files, a **System** file contains information needed to incorporate a particular kernel module into the next system configuration. General configuration information about the module type is described in the **Master** file. When the **System** component of a module's Driver Software Package (DSP) is installed, **idinstall**(1M) stores the module's **System** file information in **/etc/conf/sdevice.d/***module-name*, where the file *module-name* is the name of the module being installed. Package scripts should never access **System** files directly; only the **idinstall** and **idcheck**(1M) commands should be used.

System files typically contain data in the following format:

> **$version** *version-number*
> **$loadable** *module-name*
> *module-name configure unit ipl itype ivec sioa eioa scma ecma dmachan*

Blank lines and lines beginning with '**#**' or '*****' are considered comments and are ignored.

The first two entries are described as follows:

$version — If present in the file, this line must appear as the first non-comment line. The line specifies the version number of the **System** file format. The **System** file format being described here is version 1. If this line is omitted, version 0 (the old **sdevice** file format) is assumed.

$loadable — Indicates that the module is to be configured as a loadable module [see **modadmin**(1M)].

This line is used for dynamically loadable kernel modules only. When the line is present in the file:

> The *module-name* specified on the **$loadable** line must match the *module-name* specified in the first field of each instance line of this file.
>
> At least one instance of the module must specify the value "**Y**" in the *configure* field.
>
> The module must be coded in loadable form (requires creation of special initialization "wrapper" code for loadable modules).

To statically link a loadable module to the base kernel, the module's **$loadable** line should be commented out by inserting the character **#** in column one.

The third type of entry in the **System** file contains configuration information for each instance of the module. For example, if two instances of a device were to be configured, the device would require two lines of **System** file definitions.

The third entry type contains the following 11 fields. Each field must be delimited by white space and specify a value. Note that, except for the first two fields (*module-name* and *configure*), the remaining fields on this line are used for hardware-related modules only. That is, these fields apply to modules that have **b**, **c**, or **h** *characteristics* flags set in their **Master** files (one exception to this is **exec** modules, described below). In cases where a field does not apply to the module—regardless of the module type—the unused field must contain the value `0` (an unused *dmachan* field must contain the value `-1`).

module-name Identifies the internal name of the module. The field value must match the module name specified in the module's **Master** file.

configure Indicates (**Y** or **N**) whether **idbuild**(1M) should configure this instance of the module into the system. Note that this field can be used to configure statically linked modules or to configure dynamically loadable modules.

unit This field can be used to encode an arbitrary, module-dependent numeric value. The field is typically used to specify the number of subdevices attached to a controller or pseudo device. If this field is not used it should contain the value `0`.

ipl This field specifies the interrupt priority level for the device controlled by this module, and the priority at which the module's interrupt handler will run. Valid values are `1` (lowest priority) to `7` (highest priority); priority `8` is reserved for the clock tick interrupt. If the module is not a hardware module or does not have an interrupt handler, this field should contain the value `0`.

itype Indicates the type of interrupt sharing (if any) this hardware module supports. Note that if a module supports a number of interrupt schemes, it will require multiple system lines, with each line specifying a different *itype* field value.

Valid values are:

`0` This instance of the device does not use interrupts.

`1` This instance of the device uses an interrupt vector which cannot be shared, not even with another instance of the same module.

`2` This instance of the device uses an interrupt vector which can be shared with another instance of the same module, but can not be shared with other modules.

`3` This instance of the device uses an interrupt vector which can be shared with any instance of any hardware modules.

`4`	This instance of the device uses an EISA level-sensitive interrupt vector which can be shared with any instance of any hardware module.

If this field is not used it should contain the value `0`.

vector	Specifies the interrupt vector number used by this instance of the device. Valid values are a decimal number from `0` through the value of the highest interrupt vector number supported by the system. If the *itype* field specifies `0` (no interrupts used), this field should also specify `0`. Note that more than one device can share an interrupt vector number if the devices use the same *itype* interrupt, and that interrupt is of a type that can be shared. Note also that every instance of every module that shares an interrupt vector number must specify the same **`ipl`** values. If this field is not used it should contain the value `0`.
sioa	The start I/O address field. Specifies the lowest I/O port address through which the device communicates. Valid values are a hexadecimal number from `0` through **`FFFF`**. For non-hardware modules or devices without I/O ports, this field should contain the value `0`.
eioa	The end I/O address field. Specifies the highest (inclusive) I/O port address through which the device communicates. Valid values are a hexadecimal number from `1` through **`FFFF`**. Note that the value of the *eioa* field must be greater than or equal to the value of the *sioa* field. For non-hardware modules or devices without I/O ports, this field should contain the value `0`.
scma	The start memory controller address field. Specifies the lowest address in memory through which the device communicates. Valid values are a hexadecimal number from `10000` through **`FFFFFFFF`**. For non-hardware drivers or devices without controller memory, this field should contain the value `0`.
ecma	The end memory controller address field. Specifies the highest (inclusive) address in memory through which the device communicates. Valid values are a hexadecimal number from `10000` through **`FFFFFFFF`**. Note that the value of the *ecma* field must be greater than or equal to the value of the *scma* field. For non-hardware modules or devices without controller memory, this field should contain the value `0`.
dmachan	For hardware modules that use DMA channels, this field specifies the DMA channel number. Valid values are a decimal number from `0` through `7`. For non-hardware modules or devices that don't use DMA, this field should contain the value `-1`.

NOTES

Specifying exec Modules

For **exec** object-specific modules used to support various executable file formats (modules with **e** *characteristic* flags set in their **Master** files, two **System** file fields have special meanings:

> The *unit* field specifies the number of magic numbers supported by the module. Magic numbers are the first two bytes of an executable file, which are used to dispatch to the appropriate exec module. The module is responsible for defining an array of type **short** to hold the magic numbers. This array must be called *xxx***magic**, where *xxx* is the module's *prefix*, as defined in the **Master**(4) file. All of the **execsw** entries are processed in order, according to the *order* field of the **Master** file, calling the appropriate handlers if the magic number matches, until one returns 0 for success or an error other than **ENOEXEC**.
>
> The *itype* field controls whether a wild card **execsw** entry should be generated for the module. If *itype* is non-zero, an additional entry will be generated for the module, following the entries for the explicit magic numbers. This additional entry will have a NULL magic number pointer. This tells the system to call the handler no matter what the magic number is.

Compatibility Considerations

For compatibility with existing add-on DSP packages, **idinstall** also accepts the old (version 0) **sdevice** file format, which it converts to version 1 format before installing the file in **/etc/conf/sdevice.d**. Since the version 1 **System** file format now includes the *dmachan* field that formerly appeared in the version 0 **mdevice** file, version 0 **sdevice** files and version 0 **mdevice** files must be installed together, using a single invocation of the **idinstall** command. This allows **idinstall** to move the *dmachan* field from the module's **mdevice** file to its **System** file during conversion.

Because cross-dependencies exist in the version 0 **mdevice** and **sdevice** files for exec modules, **idinstall** cannot convert these files to version 1 files. They must be converted manually before using **idinstall**.

Note that **idinstall** also accepts obsolete **sfsys** files and converts them to version 1 **System** file format.

SEE ALSO

idbuild(1M), **idcheck**(1M), **idinstall**(1M), **Master**(4)

NAME

tc.index – configuration index file for mass-storage devices

DESCRIPTION

The **tc.index** file provides the correlation between device-specific inquiry strings and the files used by **pdimkdev**, **pdimkdtab**, and **disksetup** to control their execution. If no matching inquiry string is found in **tc.index**, generic entries found at the end of the file are used.

The generic entries should be sufficient for most types of mass-storage devices. Before spending time to generate a device-specific entry for a new mass-storage device, you should try the existing generic entry. If the generic entry works, no device-specific entry is required.

File Format

tc.index is a plain-text file consisting of keywords/value pairs. Keywords start in column 1. The value of a keyword is the rest of the text line after the keyword and any white space. A keyword/value pair must be separated by white space, either spaces or tabs. By convention, the value is separated from the keyword by a single tab. The device-specific entries in this file must be first, before any generic entries.

Keywords

The following keywords are recognized in **tc.index**:

TCINQ	Marks the beginning of a device-specific entry in the **tc.index** file. The remainder of this entry consists of the lines in the file after this keyword, up to the next **TCINQ** or **GENERIC** keyword. The inquiry string for the device in question must be duplicated character-by-character as the value of this keyword. Embedded spaces, special characters and the case of letters are all significant. The first 8 characters of the inquiry string represent the vendor identification string. The next 16 characters represent the product name. Taken together, these 24 characters are considered the inquiry string.
MKDEV	Used by **pdimkdev** and **pdimkdtab**, the name of the template file to use to create device nodes and device table entries for this device.
FORMAT	Used by **disksetup** and **diskformat**, the name of the format specification file to use when preparing this device for use by the system.
GENERIC	Marks the beginning of an entry that will be used for a given type of device if no device-specific entry is found. Generic entries are currently provided for random, sequential, CD-ROM and WORM devices. The keyword values for each of these device types are, respectively, **RANDOM**, **TAPE**, **ROM**, and **WORM**. This remainder of this entry consists of the lines in the file after this keyword up to the next **GENERIC** keyword or until the end-of-file.
#	Used to imbed comments in the **tc.index** file.

REFERENCES

diskadd(1M), **disksetup**(1M), **pdimkdev**(1M), **pdimkdtab**(1M)

NAME

term – format of compiled **term** file

SYNOPSIS

/usr/lib/share/terminfo

DESCRIPTION

Compiled **terminfo**(4) descriptions are placed under the directory **/usr/share/lib/terminfo**. In order to avoid a linear search of a huge UNIX system directory, a two-level scheme is used: **/usr/share/lib/terminfo/***c***/***name* where *name* is the name of the terminal, and *c* is the first character of *name*. Thus, **att4425** can be found in the file **/usr/share/lib/terminfo/a/att4425**. Synonyms for the same terminal are implemented by multiple links to the same compiled file.

The format has been chosen so that it is the same on all hardware. An 8-bit byte is assumed, but no assumptions about byte ordering or sign extension are made. Thus, these binary **terminfo** files can be transported to other hardware with 8-bit bytes.

Short integers are stored in two 8-bit bytes. The first byte contains the least significant 8 bits of the value, and the second byte contains the most significant 8 bits. (Thus, the value represented is 256**second+first*.) The value **-1** is represented by **0377,0377**, and the value **-2** is represented by **0376,0377**; other negative values are invalid. The **-1** generally means that a capability is missing from this terminal. The **-2** means that the capability has been canceled in the **terminfo** source and also is to be considered missing.

The compiled file is created from the source file descriptions of the terminals (see the **-I** option of **infocmp**) by using the **terminfo** compiler, **tic**, and read by the routine **setupterm** [see **curses**(3curses).] The file is divided into six parts in the following order: the header, terminal names, boolean flags, numbers, strings, and string table.

The header section begins the file. This section contains six short integers in the format described below. These integers are (1) the magic number (octal **0432**); (2) the size, in bytes, of the names section; (3) the number of bytes in the boolean section; (4) the number of short integers in the numbers section; (5) the number of offsets (short integers) in the strings section; (6) the size, in bytes, of the string table.

The terminal names section comes next. It contains the first line of the **terminfo** description, listing the various names for the terminal, separated by the bar (|) character (see **term**(5)). The section is terminated with an ASCII NUL character.

The boolean flags have one byte for each flag. This byte is either **0** or **1** as the flag is present or absent. The value of **2** means that the flag has been canceled. The capabilities are in the same order as the file <**term.h**>.

Between the boolean section and the number section, a null byte is inserted, if necessary, to ensure that the number section begins on an even byte offset. All short integers are aligned on a short word boundary.

The numbers section is similar to the boolean flags section. Each capability takes up two bytes, and is stored as a short integer. If the value represented is **-1** or **-2**, the capability is taken to be missing.

The strings section is also similar. Each capability is stored as a short integer, in the format above. A value of **-1** or **-2** means the capability is missing. Otherwise, the value is taken as an offset from the beginning of the string table. Special characters in ^X or \c notation are stored in their interpreted form, not the printing representation. Padding information ($<nn>) and parameter information (%x) are stored intact in uninterpreted form.

The final section is the string table. It contains all the values of string capabilities referenced in the string section. Each string is null terminated.

Note that it is possible for **setupterm** to expect a different set of capabilities than are actually present in the file. Either the database may have been updated since **setupterm** has been recompiled (resulting in extra unrecognized entries in the file) or the program may have been recompiled more recently than the database was updated (resulting in missing entries). The routine **setupterm** must be prepared for both possibilities—this is why the numbers and sizes are included. Also, new capabilities must always be added at the end of the lists of boolean, number, and string capabilities.

As an example, here is terminal information on the AT&T Model 37 KSR terminal as output by the **infocmp -I tty37** command:

```
37|tty37|AT&T model 37 teletype,
    hc, os, xon,
    bel=^G, cr=\r, cub1=\b, cud1=\n, cuu1=\E7, hd=\E9,
    hu=\E8, ind=\n,
```

And here is an octal dump of the **term** file, produced by the **od -c /usr/share/lib/terminfo/t/tty37** command:

```
0000000 032 001      \0 032  \0 013  \0 021 001   3  \0   3   7   |   t
0000020   t   y   3   7   |   A   T   &   T       m   o   d   e   l    
0000040   3   7       t   e   l   e   t   y   p   e  \0  \0  \0  \0  \0
0000060  \0  \0  \0 001  \0  \0  \0  \0  \0  \0  \0 001  \0  \0  \0  \0
0000100 001  \0  \0  \0  \0  \0 377 377 377 377 377 377 377 377 377 377
0000120 377 377 377 377 377 377 377 377 377 377 377 377 377 377   &  \0
0000140      \0 377 377 377 377 377 377 377 377 377 377 377 377 377 377
0000160 377 377   "  \0 377 377 377 377   (  \0 377 377 377 377 377 377
0000200 377 377   0  \0 377 377 377 377 377 377 377 377   -  \0 377 377
0000220 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
0000520 377 377 377 377 377 377 377 377 377 377 377 377 377 377   $  \0
0000540 377 377 377 377 377 377 377 377 377 377 377 377 377 377   *  \0
0000560 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
```

```
0001160 377 377 377 377 377 377 377 377 377 377 377 377 377 377   3   7
0001200   |   t   t   y   3   7   |   A   T   &   T       m   o   d   e
0001220   1       3   7       t   e   l   e   t   y   p   e  \0  \r  \0
0001240  \n  \0  \n  \0 007  \0  \b  \0 033   8  \0 033   9  \0 033   7
0001260  \0  \0
0001261
```

Some limitations: total compiled entries cannot exceed 4096 bytes; all entries in the name field cannot exceed 128 bytes.

FILES

/usr/lib/share/terminfo	compiled terminal description database
/usr/include/term.h	**terminfo** header file

SEE ALSO

curses(3curses), **infocmp**(1M), **term**(5), **terminfo**(4)

NAME

terminfo – terminal capability data base

SYNOPSIS

/usr/share/lib/terminfo/?/*

DESCRIPTION

terminfo is a database produced by **tic** that describes the capabilities of devices such as terminals and printers. Devices are described in **terminfo** source files by specifying a set of capabilities, by quantifying certain aspects of the device, and by specifying character sequences that effect particular results. This database is often used by screen oriented applications such as **vi** and **curses** programs, as well as by some UNIX system commands such as **ls** and **more**. This usage allows them to work with a variety of devices without changes to the programs.

terminfo source files consist of one or more device descriptions. Each description consists of a header (beginning in column 1) and one or more lines that list the features for that particular device. Every line in a **terminfo** source file must end in a comma (**,**). Every line in a **terminfo** source file except the header must be indented with one or more white spaces (either spaces or tabs).

Entries in **terminfo** source files consist of a number of comma-separated fields. White space after each comma is ignored. Embedded commas must be escaped by using a backslash. The following example shows the format of a **terminfo** source file.

> *alias*$_1$ | *alias*$_2$ | ... | *alias*$_n$ | *longname,*
> *<white space>* **am, lines #24,**
> *<white space>* **home=\Eeh,**

The first line, commonly referred to as the header line, must begin in column one and must contain at least two aliases separated by vertical bars. The last field in the header line must be the long name of the device and it may contain any string. Alias names must be unique in the **terminfo** database and they must conform to UNIX system file naming conventions [see **tic**(1M)]; they cannot, for example, contain white space or slashes.

Every device must be assigned a name, such as "vt100". Device names (except the long name) should be chosen using the following conventions. The name should not contain hyphens because hyphens are reserved for use when adding suffixes that indicate special modes.

These special modes may be modes that the hardware can be in, or user preferences. To assign a special mode to a particular device, append a suffix consisting of a hyphen and an indicator of the mode to the device name. For example, the **-w** suffix means "wide mode"; when specified, it allows for a width of 132 columns instead of the standard 80 columns. Therefore, if you want to use a vt100 device set to wide mode, name the device "vt100-w." Use the following suffixes where possible.

Suffix	Meaning	Example
-w	Wide mode (more than 80 columns)	**5410-w**
-am	With auto. margins (usually default)	**vt100-am**
-nam	Without automatic margins	**vt100-nam**
-*n*	Number of lines on the screen	**2300-40**
-na	No arrow keys (leave them in local)	**c100-na**
-*n*p	Number of pages of memory	**c100-4p**
-rv	Reverse video	**4415-rv**

The **terminfo** reference manual page is organized in two sections: "DEVICE CAPABILITIES" and "PRINTER CAPABILITIES."

PART 1: DEVICE CAPABILITIES

Capabilities in **terminfo** are of three types: Boolean capabilities (which show that a device has or does not have a particular feature), numeric capabilities (which quantify particular features of a device), and string capabilities (which provide sequences that can be used to perform particular operations on devices).

In the following table, a Variable is the name by which a **C** programmer accesses a capability (at the **terminfo** level). A Capname is the short name for a capability specified in the **terminfo** source file. It is used by a person updating the source file and by the **tput** command. A Termcap Code is a two-letter sequence that corresponds to the **termcap** capability name. (Note that **termcap** is no longer supported.)

Capability names have no real length limit, but an informal limit of five characters has been adopted to keep them short. Whenever possible, capability names are chosen to be the same as or similar to those specified by the ANSI X3.64-1979 standard. Semantics are also intended to match those of the ANSI standard.

All string capabilities listed below may have padding specified, with the exception of those used for input. Input capabilities, listed under the **Strings** section in the following tables, have names beginning with **key_**. The **#i** symbol in the description field of the following tables refers to the *i*th parameter.

Booleans

Variable	Cap-name	Termcap Code	Description
auto_left_margin	bw	bw	cub1 wraps from column 0 to last column
auto_right_margin	am	am	Terminal has automatic margins
back_color_erase	bce	be	Screen erased with background color
can_change	ccc	cc	Terminal can re-define existing color
ceol_standout_glitch	xhp	xs	Standout not erased by overwriting (hp)
col_addr_glitch	xhpa	YA	Only positive motion for hpa/mhpa caps
cpi_changes_res	cpix	YF	Changing character pitch changes resolution
cr_cancels_micro_mode	crxm	YB	Using cr turns off micro mode

Variable	Cap-name	Termcap Code	Description
eat_newline_glitch	xenl	xn	Newline ignored after 80 columns (Concept)
erase_overstrike	eo	eo	Can erase overstrikes with a blank
generic_type	gn	gn	Generic line type (*e.g.*, dialup, switch)
hard_copy	hc	hc	Hardcopy terminal
hard_cursor	chts	HC	Cursor is hard to see
has_meta_key	km	km	Has a meta key (shift, sets parity bit)
has_print_wheel	daisy	YC	Printer needs operator to change character set
has_status_line	hs	hs	Has extra "status line"
hue_lightness_saturation	hls	hl	Terminal uses only HLS color notation (Tektronix)
insert_null_glitch	in	in	Insert mode distinguishes nulls
lpi_changes_res	lpix	YG	Changing line pitch changes resolution
memory_above	da	da	Display may be retained above the screen
memory_below	db	db	Display may be retained below the screen
move_insert_mode	mir	mi	Safe to move while in insert mode
move_standout_mode	msgr	ms	Safe to move in standout modes
needs_xon_xoff	nxon	nx	Padding won't work, xon/xoff required
no_esc_ctlc	xsb	xb	Beehive (f1=escape, f2=ctrl C)
non_rev_rmcup	nrrmc	NR	**smcup** does not reverse **rmcup**
no_pad_char	npc	NP	Pad character doesn't exist
over_strike	os	os	Terminal overstrikes on hard-copy terminal
prtr_silent	mc5i	5i	Printer won't echo on screen
row_addr_glitch	xvpa	YD	Only positive motion for **vpa**/**mvpa** caps
semi_auto_right_margin	sam	YE	Printing in last column causes **cr**
status_line_esc_ok	eslok	es	Escape can be used on the status line
dest_tabs_magic_smso	xt	xt	Destructive tabs, magic **smso** char (t1061)
tilde_glitch	hz	hz	Hazeltine; can't print tilde (~)
transparent_underline	ul	ul	Underline character overstrikes
xon_xoff	xon	xo	Terminal uses xon/xoff handshaking

Numbers

Variable	Cap-name	Termcap Code	Description
buffer_capacity	bufsz	Ya	Number of bytes buffered before printing
buttons	btns	BT	Number of buttons on the mouse
columns	cols	co	Number of columns in a line
dot_vert_spacing	spinv	Yb	Spacing of pins vertically in pins per inch
dot_horz_spacing	spinh	Yc	Spacing of dots horizontally in dots per inch
init_tabs	it	it	Tabs initially every # spaces
label_height	lh	lh	Number of rows in each label
label_width	lw	lw	Number of columns in each label

Variable	Cap-name	Termcap Code	Description
`lines`	`lines`	`li`	Number of lines on a screen or a page
`lines_of_memory`	`lm`	`lm`	Lines of memory if > `lines`; `0` means varies
`magic_cookie_glitch`	`xmc`	`sg`	Number of blank characters left by `smso` or `rmso`
`max_colors`	`colors`	`Co`	Maximum number of colors on the screen
`max_micro_address`	`maddr`	`Yd`	Maximum value in `micro_..._address`
`max_micro_jump`	`mjump`	`Ye`	Maximum value in `parm_..._micro`
`max_pairs`	`pairs`	`pa`	Maximum number of color-pairs on the screen
`micro_col_size`	`mcs`	`Yf`	Character step size when in micro mode
`micro_line_size`	`mls`	`Yg`	Line step size when in micro mode
`no_color_video`	`ncv`	`NC`	Video attributes that can't be used with colors
`number_of_pins`	`npins`	`Yh`	Number of pins in print-head
`num_labels`	`nlab`	`Nl`	Number of labels on screen (start at 1)
`output_res_char`	`orc`	`Yi`	Horizontal resolution in units per character
`output_res_line`	`orl`	`Yj`	Vertical resolution in units per line
`output_res_horz_inch`	`orhi`	`Yk`	Horizontal resolution in units per inch
`output_res_vert_inch`	`orvi`	`Yl`	Vertical resolution in units per inch
`padding_baud_rate`	`pb`	`pb`	Lowest baud rate where padding needed
`virtual_terminal`	`vt`	`vt`	Virtual terminal number (UNIX system)
`wide_char_size`	`widcs`	`Yn`	Character step size when in double wide mode
`width_status_line`	`wsl`	`ws`	Number of columns in status line

Strings

Variable	Cap-name	Termcap Code	Description
`acs_chars`	`acsc`	`ac`	Graphic charset pairs aAbBcC
`alt_scancode_esc`	`scesca`	`S8`	Alternate escape for scancode emulation (default is for vt100)
`back_tab`	`cbt`	`bt`	Back tab
`bell`	`bel`	`bl`	Audible signal (bell)
`bit_image_repeat`	`birep`	`Zy`	Repeat bit-image cell #1 #2 times (use `tparm`)
`bit_image_newline`	`binel`	`Zz`	Move to next row of the bit image (use `tparm`)
`bit_image_carriage_return`	`bicr`	`Yv`	Move to beginning of same row (use `tparm`)
`carriage_return`	`cr`	`cr`	Carriage return
`change_char_pitch`	`cpi`	`ZA`	Change number of characters per inch
`change_line_pitch`	`lpi`	`ZB`	Change number of lines per inch
`change_res_horz`	`chr`	`ZC`	Change horizontal resolution
`change_res_vert`	`cvr`	`ZD`	Change vertical resolution
`change_scroll_region`	`csr`	`cs`	Change to lines #1 through #2 (vt100)
`char_padding`	`rmp`	`rP`	Like `ip` but when in replace mode
`char_set_names`	`csnm`	`Zy`	List of character set names

Variable	Cap-name	Termcap Code	Description
clear_all_tabs	tbc	ct	Clear all tab stops
clear_margins	mgc	MC	Clear all margins (top, bottom, and sides)
clear_screen	clear	cl	Clear screen and home cursor
clr_bol	el1	cb	Clear to beginning of line, inclusive
clr_eol	el	ce	Clear to end of line
clr_eos	ed	cd	Clear to end of display
code_set_init	csin	ci	Init sequence for multiple codesets
color_names	colornm	Yw	Give name for color #1
column_address	hpa	ch	Horizontal position absolute
command_character	cmdch	CC	Terminal settable cmd character in prototype
cursor_address	cup	cm	Move to row #1 col #2
cursor_down	cud1	do	Down one line
cursor_home	home	ho	Home cursor (if no **cup**)
cursor_invisible	civis	vi	Make cursor invisible
cursor_left	cub1	le	Move left one space.
cursor_mem_address	mrcup	CM	Memory relative cursor addressing
cursor_normal	cnorm	ve	Make cursor appear normal (undo **vs/vi**)
cursor_right	cuf1	nd	Non-destructive space (cursor or carriage right)
cursor_to_ll	ll	ll	Last line, first column (if no **cup**)
cursor_up	cuu1	up	Upline (cursor up)
cursor_visible	cvvis	vs	Make cursor very visible
define_bit_image_region	defbi	Yx	Define rectangular bit-image region (use **tparm**)
define_char	defc	ZE	Define a character in a character set †
delete_character	dch1	dc	Delete character
delete_line	dl1	dl	Delete line
device_type	devt	dv	Indicate language/codeset support
dis_status_line	dsl	ds	Disable status line
display_pc_char	dispc	S1	Display PC character
down_half_line	hd	hd	Half-line down (forward 1/2 linefeed)
ena_acs	enacs	eA	Enable alternate character set
end_bit_image_region	endbi	Yy	End a bit-image region (use **tparm**)
enter_alt_charset_mode	smacs	as	Start alternate character set
enter_am_mode	smam	SA	Turn on automatic margins
enter_blink_mode	blink	mb	Turn on blinking
enter_bold_mode	bold	md	Turn on bold (extra bright) mode
enter_ca_mode	smcup	ti	String to begin programs that use **cup**
enter_delete_mode	smdc	dm	Delete mode (enter)
enter_dim_mode	dim	mh	Turn on half-bright mode
enter_doublewide_mode	swidm	ZF	Enable double wide printing
enter_draft_quality	sdrfq	ZG	Set draft quality print

Variable	Cap-name	Termcap Code	Description
enter_insert_mode	smir	im	Insert mode (enter)
enter_italics_mode	sitm	ZH	Enable italics
enter_leftward_mode	slm	ZI	Enable leftward carriage motion
enter_micro_mode	smicm	ZJ	Enable micro motion capabilities
enter_near_letter_quality	snlq	ZK	Set near-letter quality print
enter_normal_quality	snrmq	ZL	Set normal quality print
enter_pc_charset_mode	smpch	S2	Enter PC character display mode
enter_protected_mode	prot	mp	Turn on protected mode
enter_reverse_mode	rev	mr	Turn on reverse video mode
enter_scancode_mode	smsc	S4	Enter PC scancode mode
enter_secure_mode	invis	mk	Turn on blank mode (characters invisible)
enter_shadow_mode	sshm	ZM	Enable shadow printing
enter_standout_mode	smso	so	Begin standout mode
enter_subscript_mode	ssubm	ZN	Enable subscript printing
enter_superscript_mode	ssupm	ZO	Enable superscript printing
enter_underline_mode	smul	us	Start underscore mode
enter_upward_mode	sum	ZP	Enable upward carriage motion
enter_xon_mode	smxon	SX	Turn on xon/xoff handshaking
erase_chars	ech	ec	Erase #1 characters
exit_alt_charset_mode	rmacs	ae	End alternate character set
exit_am_mode	rmam	RA	Turn off automatic margins
exit_attribute_mode	sgr0	me	Turn off all attributes
exit_ca_mode	rmcup	te	String to end programs that use **cup**
exit_delete_mode	rmdc	ed	End delete mode
exit_doublewide_mode	rwidm	ZQ	Disable double wide printing
exit_insert_mode	rmir	ei	End insert mode
exit_italics_mode	ritm	ZR	Disable italics
exit_leftward_mode	rlm	ZS	Enable rightward (normal) carriage motion
exit_micro_mode	rmicm	ZT	Disable micro motion capabilities
exit_pc_charset_mode	rmpch	S3	Disable PC character display mode
exit_scancode_mode	rmsc	S5	Disable PC scancode mode
exit_shadow_mode	rshm	ZU	Disable shadow printing
exit_standout_mode	rmso	se	End standout mode
exit_subscript_mode	rsubm	ZV	Disable subscript printing
exit_superscript_mode	rsupm	ZW	Disable superscript printing
exit_underline_mode	rmul	ue	End underscore mode
exit_upward_mode	rum	ZX	Enable downward (normal) carriage motion
exit_xon_mode	rmxon	RX	Turn off xon/xoff handshaking
flash_screen	flash	vb	Visible bell (may not move cursor)
form_feed	ff	ff	Hardcopy terminal page eject
from_status_line	fsl	fs	Return from status line
get_mouse	getm	Gm	Curses should get button events

Variable	Cap-name	Termcap Code	Description
init_1string	**is1**	**i1**	Terminal or printer initialization string
init_2string	**is2**	**is**	Terminal or printer initialization string
init_3string	**is3**	**i3**	Terminal or printer initialization string
init_file	**if**	**if**	Name of initialization file
init_prog	**iprog**	**iP**	Path name of program for initialization
initialize_color	**initc**	**Ic**	Initialize the definition of color
initialize_pair	**initp**	**Ip**	Initialize color-pair
insert_character	**ich1**	**ic**	Insert character
insert_line	**il1**	**al**	Add new blank line
insert_padding	**ip**	**ip**	Insert pad after character inserted

The "**key_**" strings are sent by specific keys. The "**key_**" descriptions include the macro, defined in **curses.h**, for the code returned by the **curses** routine **getch** when the key is pressed [see **curs_getch**(3curses)].

key_a1	**ka1**	**K1**	**KEY_A1**, upper left of keypad
key_a3	**ka3**	**K3**	**KEY_A3**, upper right of keypad
key_b2	**kb2**	**K2**	**KEY_B2**, center of keypad
key_backspace	**kbs**	**kb**	**KEY_BACKSPACE**, sent by backspace key
key_beg	**kbeg**	**@1**	**KEY_BEG**, sent by beg(inning) key
key_btab	**kcbt**	**kB**	**KEY_BTAB**, sent by back-tab key
key_c1	**kc1**	**K4**	**KEY_C1**, lower left of keypad
key_c3	**kc3**	**K5**	**KEY_C3**, lower right of keypad
key_cancel	**kcan**	**@2**	**KEY_CANCEL**, sent by cancel key
key_catab	**ktbc**	**ka**	**KEY_CATAB**, sent by clear-all-tabs key
key_clear	**kclr**	**kC**	**KEY_CLEAR**, sent by clear-screen or erase key
key_close	**kclo**	**@3**	**KEY_CLOSE**, sent by close key
key_command	**kcmd**	**@4**	**KEY_COMMAND**, sent by cmd (command) key
key_copy	**kcpy**	**@5**	**KEY_COPY**, sent by copy key
key_create	**kcrt**	**@6**	**KEY_CREATE**, sent by create key
key_ctab	**kctab**	**kt**	**KEY_CTAB**, sent by clear-tab key
key_dc	**kdch1**	**kD**	**KEY_DC**, sent by delete-character key
key_dl	**kdl1**	**kL**	**KEY_DL**, sent by delete-line key
key_down	**kcud1**	**kd**	**KEY_DOWN**, sent by terminal down-arrow key
key_eic	**krmir**	**kM**	**KEY_EIC**, sent by **rmir** or **smir** in insert mode
key_end	**kend**	**@7**	**KEY_END**, sent by end key
key_enter	**kent**	**@8**	**KEY_ENTER**, sent by enter/send key
key_eol	**kel**	**kE**	**KEY_EOL**, sent by clear-to-end-of-line key
key_eos	**ked**	**kS**	**KEY_EOS**, sent by clear-to-end-of-screen key

Variable	Cap-name	Termcap Code	Description
key_exit	**kext**	**@9**	**KEY_EXIT**, sent by exit key
key_f0	**kf0**	**k0**	**KEY_F(0)**, sent by function key f0
key_f1	**kf1**	**k1**	**KEY_F(1)**, sent by function key f1
key_f2	**kf2**	**k2**	**KEY_F(2)**, sent by function key f2
key_f3	**kf3**	**k3**	**KEY_F(3)**, sent by function key f3
key_f4	**kf4**	**k4**	**KEY_F(4)**, sent by function key f4
key_f5	**kf5**	**k5**	**KEY_F(5)**, sent by function key f5
key_f6	**kf6**	**k6**	**KEY_F(6)**, sent by function key f6
key_f7	**kf7**	**k7**	**KEY_F(7)**, sent by function key f7
key_f8	**kf8**	**k8**	**KEY_F(8)**, sent by function key f8
key_f9	**kf9**	**k9**	**KEY_F(9)**, sent by function key f9
key_f10	**kf10**	**k;**	**KEY_F(10)**, sent by function key f10
key_f11	**kf11**	**F1**	**KEY_F(11)**, sent by function key f11
key_f12	**kf12**	**F2**	**KEY_F(12)**, sent by function key f12
key_f13	**kf13**	**F3**	**KEY_F(13)**, sent by function key f13
key_f14	**kf14**	**F4**	**KEY_F(14)**, sent by function key f14
key_f15	**kf15**	**F5**	**KEY_F(15)**, sent by function key f15
key_f16	**kf16**	**F6**	**KEY_F(16)**, sent by function key f16
key_f17	**kf17**	**F7**	**KEY_F(17)**, sent by function key f17
key_f18	**kf18**	**F8**	**KEY_F(18)**, sent by function key f18
key_f19	**kf19**	**F9**	**KEY_F(19)**, sent by function key f19
key_f20	**kf20**	**FA**	**KEY_F(20)**, sent by function key f20
key_f21	**kf21**	**FB**	**KEY_F(21)**, sent by function key f21
key_f22	**kf22**	**FC**	**KEY_F(22)**, sent by function key f22
key_f23	**kf23**	**FD**	**KEY_F(23)**, sent by function key f23
key_f24	**kf24**	**FE**	**KEY_F(24)**, sent by function key f24
key_f25	**kf25**	**FF**	**KEY_F(25)**, sent by function key f25
key_f26	**kf26**	**FG**	**KEY_F(26)**, sent by function key f26
key_f27	**kf27**	**FH**	**KEY_F(27)**, sent by function key f27
key_f28	**kf28**	**FI**	**KEY_F(28)**, sent by function key f28
key_f29	**kf29**	**FJ**	**KEY_F(29)**, sent by function key f29
key_f30	**kf30**	**FK**	**KEY_F(30)**, sent by function key f30
key_f31	**kf31**	**FL**	**KEY_F(31)**, sent by function key f31
key_f32	**kf32**	**FM**	**KEY_F(32)**, sent by function key f32
key_f33	**kf33**	**FN**	**KEY_F(13)**, sent by function key f13
key_f34	**kf34**	**FO**	**KEY_F(34)**, sent by function key f34
key_f35	**kf35**	**FP**	**KEY_F(35)**, sent by function key f35
key_f36	**kf36**	**FQ**	**KEY_F(36)**, sent by function key f36
key_f37	**kf37**	**FR**	**KEY_F(37)**, sent by function key f37
key_f38	**kf38**	**FS**	**KEY_F(38)**, sent by function key f38
key_f39	**kf39**	**FT**	**KEY_F(39)**, sent by function key f39
key_f40	**kf40**	**FU**	**KEY_F(40)**, sent by function key f40
key_f41	**kf41**	**FV**	**KEY_F(41)**, sent by function key f41
key_f42	**kf42**	**FW**	**KEY_F(42)**, sent by function key f42
key_f43	**kf43**	**FX**	**KEY_F(43)**, sent by function key f43

Variable	Cap-name	Termcap Code	Description
key_f44	kf44	FY	KEY_F(44), sent by function key f44
key_f45	kf45	FZ	KEY_F(45), sent by function key f45
key_f46	kf46	Fa	KEY_F(46), sent by function key f46
key_f47	kf47	Fb	KEY_F(47), sent by function key f47
key_f48	kf48	Fc	KEY_F(48), sent by function key f48
key_f49	kf49	Fd	KEY_F(49), sent by function key f49
key_f50	kf50	Fe	KEY_F(50), sent by function key f50
key_f51	kf51	Ff	KEY_F(51), sent by function key f51
key_f52	kf52	Fg	KEY_F(52), sent by function key f52
key_f53	kf53	Fh	KEY_F(53), sent by function key f53
key_f54	kf54	Fi	KEY_F(54), sent by function key f54
key_f55	kf55	Fj	KEY_F(55), sent by function key f55
key_f56	kf56	Fk	KEY_F(56), sent by function key f56
key_f57	kf57	Fl	KEY_F(57), sent by function key f57
key_f58	kf58	Fm	KEY_F(58), sent by function key f58
key_f59	kf59	Fn	KEY_F(59), sent by function key f59
key_f60	kf60	Fo	KEY_F(60), sent by function key f60
key_f61	kf61	Fp	KEY_F(61), sent by function key f61
key_f62	kf62	Fq	KEY_F(62), sent by function key f62
key_f63	kf63	Fr	KEY_F(63), sent by function key f63
key_find	kfnd	@0	KEY_FIND, sent by find key
key_help	khlp	%1	KEY_HELP, sent by help key
key_home	khome	kh	KEY_HOME, sent by home key
key_ic	kich1	kI	KEY_IC, sent by ins-char/enter ins-mode key
key_il	kil1	kA	KEY_IL, sent by insert-line key
key_left	kcub1	kl	KEY_LEFT, sent by terminal left-arrow key
key_ll	kll	kH	KEY_LL, sent by home-down key
key_mark	kmrk	%2	KEY_MARK, sent by mark key
key_message	kmsg	%3	KEY_MESSAGE, sent by message key
key_mouse	kmous	Km	Mouse event has occurred
key_move	kmov	%4	KEY_MOVE, sent by move key
key_next	knxt	%5	KEY_NEXT, sent by next-object key
key_npage	knp	kN	KEY_NPAGE, sent by next-page key
key_open	kopn	%6	KEY_OPEN, sent by open key
key_options	kopt	%7	KEY_OPTIONS, sent by options key
key_ppage	kpp	kP	KEY_PPAGE, sent by previous-page key
key_previous	kprv	%8	KEY_PREVIOUS, sent by previous-object key
key_print	kprt	%9	KEY_PRINT, sent by print or copy key
key_redo	krdo	%0	KEY_REDO, sent by redo key
key_reference	kref	&1	KEY_REFERENCE, sent by ref(erence) key
key_refresh	krfr	&2	KEY_REFRESH, sent by refresh key
key_replace	krpl	&3	KEY_REPLACE, sent by replace key

Variable	Cap-name	Termcap Code	Description
key_restart	krst	&4	KEY_RESTART, sent by restart key
key_resume	kres	&5	KEY_RESUME, sent by resume key
key_right	kcuf1	kr	KEY_RIGHT, sent by terminal right-arrow key
key_save	ksav	&6	KEY_SAVE, sent by save key
key_sbeg	kBEG	&9	KEY_SBEG, sent by shifted beginning key
key_scancel	kCAN	&0	KEY_SCANCEL, sent by shifted cancel key
key_scommand	kCMD	*1	KEY_SCOMMAND, sent by shifted command key
key_scopy	kCPY	*2	KEY_SCOPY, sent by shifted copy key
key_screate	kCRT	*3	KEY_SCREATE, sent by shifted create key
key_sdc	kDC	*4	KEY_SDC, sent by shifted delete-char key
key_sdl	kDL	*5	KEY_SDL, sent by shifted delete-line key
key_select	kslt	*6	KEY_SELECT, sent by select key
key_send	kEND	*7	KEY_SEND, sent by shifted end key
key_seol	kEOL	*8	KEY_SEOL, sent by shifted clear-line key
key_sexit	kEXT	*9	KEY_SEXIT, sent by shifted exit key
key_sf	kind	kF	KEY_SF, sent by scroll-forward/down key
key_sfind	kFND	*0	KEY_SFIND, sent by shifted find key
key_shelp	kHLP	#1	KEY_SHELP, sent by shifted help key
key_shome	kHOM	#2	KEY_SHOME, sent by shifted home key
key_sic	kIC	#3	KEY_SIC, sent by shifted input key
key_sleft	kLFT	#4	KEY_SLEFT, sent by shifted left-arrow key
key_smessage	kMSG	%a	KEY_SMESSAGE, sent by shifted message key
key_smove	kMOV	%b	KEY_SMOVE, sent by shifted move key
key_snext	kNXT	%c	KEY_SNEXT, sent by shifted next key
key_soptions	kOPT	%d	KEY_SOPTIONS, sent by shifted options key
key_sprevious	kPRV	%e	KEY_SPREVIOUS, sent by shifted prev key
key_sprint	kPRT	%f	KEY_SPRINT, sent by shifted print key
key_sr	kri	kR	KEY_SR, sent by scroll-backward/up key
key_sredo	kRDO	%g	KEY_SREDO, sent by shifted redo key
key_sreplace	kRPL	%h	KEY_SREPLACE, sent by shifted replace key
key_sright	kRIT	%i	KEY_SRIGHT, sent by shifted right-arrow key
key_srsume	kRES	%j	KEY_SRSUME, sent by shifted resume key
key_ssave	kSAV	!1	KEY_SSAVE, sent by shifted save key
key_ssuspend	kSPD	!2	KEY_SSUSPEND, sent by shifted suspend

Variable	Cap-name	Termcap Code	Description
			key
key_stab	khts	kT	KEY_STAB, sent by set-tab key
key_sundo	kUND	!3	KEY_SUNDO, sent by shifted undo key
key_suspend	kspd	&7	KEY_SUSPEND, sent by suspend key
key_undo	kund	&8	KEY_UNDO, sent by undo key
key_up	kcuu1	ku	KEY_UP, sent by terminal up-arrow key
keypad_local	rmkx	ke	Out of "keypad-transmit" mode
keypad_xmit	smkx	ks	Put terminal in "keypad-transmit" mode
lab_f0	lf0	l0	Labels on function key f0 if not f0
lab_f1	lf1	l1	Labels on function key f1 if not f1
lab_f2	lf2	l2	Labels on function key f2 if not f2
lab_f3	lf3	l3	Labels on function key f3 if not f3
lab_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)
micro_column_address	mhpa	ZY	Like column_address for micro adjustment
micro_down	mcud1	ZZ	Like cursor_down for micro adjustment
micro_left	mcub1	Za	Like cursor_left for micro adjustment
micro_right	mcuf1	Zb	Like cursor_right for micro adjustment
micro_row_address	mvpa	Zc	Like row_address for micro adjustment
micro_up	mcuu1	Zd	Like cursor_up for micro adjustment
mouse_info	minfo	Mi	Mouse status information
newline	nel	nw	Newline (behaves like cr followed by lf)
order_of_pins	porder	Ze	Matches software bits to print-head pins
orig_colors	oc	oc	Set all color(-pair)s to the original ones
orig_pair	op	op	Set default color-pair to the original one
pad_char	pad	pc	Pad character (rather than null)
parm_dch	dch	DC	Delete #1 chars
parm_delete_line	dl	DL	Delete #1 lines
parm_down_cursor	cud	DO	Move down #1 lines.
parm_down_micro	mcud	Zf	Like parm_down_cursor for micro adjust.
parm_ich	ich	IC	Insert #1 blank chars

Variable	Cap-name	Termcap Code	Description
parm_index	indn	SF	Scroll forward #1 lines.
parm_insert_line	il	AL	Add #1 new blank lines
parm_left_cursor	cub	LE	Move cursor left #1 spaces
parm_left_micro	mcub	Zg	Like parm_left_cursor for micro adjust.
parm_right_cursor	cuf	RI	Move right #1 spaces.
parm_right_micro	mcuf	Zh	Like parm_right_cursor for micro adjust.
parm_rindex	rin	SR	Scroll backward #1 lines.
parm_up_cursor	cuu	UP	Move cursor up #1 lines.
parm_up_micro	mcuu	Zi	Like parm_up_cursor for micro adjust.
pc_term_options	pctrm	S6	PC terminal options
pkey_key	pfkey	pk	Prog funct key #1 to type string #2
pkey_local	pfloc	pl	Prog funct key #1 to execute string #2
pkey_plab	pfxl	xl	Prog key #1 to xmit string #2 and show string #3
pkey_xmit	pfx	px	Prog funct key #1 to xmit string #2
plab_norm	pln	pn	Prog label #1 to show string #2
print_screen	mc0	ps	Print contents of the screen
prtr_non	mc5p	pO	Turn on the printer for #1 bytes
prtr_off	mc4	pf	Turn off the printer
prtr_on	mc5	po	Turn on the printer
repeat_char	rep	rp	Repeat char #1 #2 times
req_for_input	rfi	RF	Send next input char (for ptys)
req_mouse_pos	reqmp	RQ	Request mouse position report
reset_1string	rs1	r1	Reset terminal completely to sane modes
reset_2string	rs2	r2	Reset terminal completely to sane modes
reset_3string	rs3	r3	Reset terminal completely to sane modes
reset_file	rf	rf	Name of file containing reset string
restore_cursor	rc	rc	Restore cursor to position of last sc
row_address	vpa	cv	Vertical position absolute
save_cursor	sc	sc	Save cursor position
scancode_escape	scesc	S7	Escape for scancode emulation
scroll_forward	ind	sf	Scroll text up
scroll_reverse	ri	sr	Scroll text down
select_char_set	scs	Zj	Select character set
set0_des_seq	s0ds	s0	Shift into codeset 0 (EUC set 0, ASCII)
set1_des_seq	s1ds	s1	Shift into codeset 1
set2_des_seq	s2ds	s2	Shift into codeset 2
set3_des_seq	s3ds	s3	Shift into codeset 3
set_a_background	setab	AB	Set background color using ANSI escape
set_a_foreground	setaf	AF	Set foreground color using ANSI escape
set_attributes	sgr	sa	Define the video attributes #1-#9
set_background	setb	Sb	Set current background color
set_bottom_margin	smgb	Zk	Set bottom margin at current line
set_bottom_margin_parm	smgbp	Zl	Set bottom margin at line #1 or #2

Variable	Cap-name	Termcap Code	Description
			lines from bottom
set_color_band	setcolor	Yz	Change to ribbon color #1
set_color_pair	scp	sp	Set current color-pair
set_foreground	setf	Sf	Set current foreground color1
set_left_margin	smgl	ML	Set left margin at current line
set_left_margin_parm	smglp	Zm	Set left (right) margin at column #1 (#2)
set_lr_margin	smglr	ML	Sets both left and right margins
set_page_length	slines	YZ	Set page length to #1 lines (use **tparm**)
set_pglen_inch	slength	YI	Set page length to #1 hundredths of an inch (use **tparm**)
set_right_margin	smgr	MR	Set right margin at current column
set_right_margin_parm	smgrp	Zn	Set right margin at column #1
set_tab	hts	st	Set a tab in all rows, current column
set_tb_margin	smgtb	MT	Sets both top and bottom margins
set_top_margin	smgt	Zo	Set top margin at current line
set_top_margin_parm	smgtp	Zp	Set top (bottom) margin at line #1 (#2)
set_window	wind	wi	Current window is lines #1-#2 cols #3-#4
start_bit_image	sbim	Zq	Start printing bit image graphics
start_char_set_def	scsd	Zr	Start definition of a character set
stop_bit_image	rbim	Zs	End printing bit image graphics
stop_char_set_def	rcsd	Zt	End definition of a character set
subscript_characters	subcs	Zu	List of "subscript-able" characters
superscript_characters	supcs	Zv	List of "superscript-able" characters
tab	ht	ta	Tab to next 8-space hardware tab stop
these_cause_cr	docr	Zw	Printing any of these chars causes **cr**
to_status_line	tsl	ts	Go to status line, col #1
underline_char	uc	uc	Underscore one char and move past it
up_half_line	hu	hu	Half-line up (reverse 1/2 linefeed)
xoff_character	xoffc	XF	X-off character
xon_character	xonc	XN	X-on character
zero_motion	zerom	Zx	No motion for the subsequent character

Sample Entry

The following entry, which describes the AT&T 610 terminal, is among the more complex entries in the **terminfo** file as of this writing.

```
610|610bct|ATT610|att610|AT&T 610; 80 column; 98key keyboard
   am, eslok, hs, mir, msgr, xenl, xon,
   cols#80, it#8, lh#2, lines#24, lw#8, nlab#8, wsl#80,
   acsc=''aaffggjjkkllmmnnooppqqrrssttuuvvwwxxyyzz{{||}}~~,
   bel=^G, blink=\E[5m, bold=\E[1m, cbt=\E[Z,
   civis=\E[?25l, clear=\E[H\E[J, cnorm=\E[?25h\E[?12l,
   cr=\r, csr=\E[%i%p1%d;%p2%dr, cub=\E[%p1%dD, cub1=\b,
   cud=\E[%p1%dB, cud1=\E[B, cuf=\E[%p1%dC, cuf1=\E[C,
   cup=\E[%i%p1%d;%p2%dH, cuu=\E[%p1%dA, cuu1=\E[A,
   cvvis=\E[?12;25h, dch=\E[%p1%dP, dch1=\E[P, dim=\E[2m,
   dl=\E[%p1%dM, dl1=\E[M, ed=\E[J, el=\E[K, el1=\E[1K,
   flash=\E[?5h$<200>\E[?5l, fsl=\E8, home=\E[H, ht=\t,
ich=\E[%p1%d@, il=\E[%p1%dL, il1=\E[L, ind=\ED, .ind=\ED$<9>,
   invis=\E[8m,
   is1=\E[8;0 | \E[?3;4;5;13;15l\E[13;20l\E[?7h\E[12h\E(B\E)0,
   is2=\E[0m^O, is3=\E(B\E)0, kLFT=\E[\s@, kRIT=\E[\sA,
   kbs=^H, kcbt=\E[Z, kclr=\E[2J, kcub1=\E[D, kcud1=\E[B,
   kcuf1=\E[C, kcuu1=\E[A, kf1=\EOc, kf10=\ENp,
   kf11=\ENq, kf12=\ENr, kf13=\ENs, kf14=\ENt, kf2=\EOd,
   kf3=\EOe, kf4=\EOf, kf5=\EOg, kf6=\EOh, kf7=\EOi,
   kf8=\EOj, kf9=\ENo, khome=\E[H, kind=\E[S, kri=\E[T,
   ll=\E[24H, mc4=\E[?4i, mc5=\E[?5i, nel=\EE,
   pfxl=\E[%p1%d;%p2%l%02dq%?%p1%{9}%<%t\s\s\sF%p1%1d\s\s\s\s\s
\s\s\s\s\s\s%;%p2%s,
   pln=\E[%p1%d;0;0;0q%p2%:-16.16s, rc=\E8, rev=\E[7m,
   ri=\EM, rmacs=^O, rmir=\E[4l, rmln=\E[2p, rmso=\E[m,
   rmul=\E[m, rs2=\Ec\E[?3l, sc=\E7,
   sgr=\E[0%?%p6%t;1%;%?%p5%t;2%;%?%p2%t;4%;%?%p4%t;5%;
%?%p3%p1% | %t;7%;%?%p7%t;8%;m%?%p9%t^N%e^O%;,
   sgr0=\E[m^O, smacs=^N, smir=\E[4h, smln=\E[p,
   smso=\E[7m, smul=\E[4m, tsl=\E7\E[25;%i%p1%dx,
```

Types of Capabilities in the Sample Entry

The sample entry shows the formats for the three types of **terminfo** capabilities listed: Boolean, numeric, and string. All capabilities specified in the **terminfo** source file must be followed by commas, including the last capability in the source file. In **terminfo** source files, capabilities are referenced by their capability names (as shown in the previous tables).

Boolean capabilities are specified simply by their comma separated cap names.

Numeric capabilities are followed by the character '#' and then a positive integer value. Thus, in the sample, **cols** (which shows the number of columns available on a device) is assigned the value **80** for the AT&T 610. (Values for numeric capabilities may be specified in decimal, octal, or hexadecimal, using normal C programming language conventions.)

Finally, string-valued capabilities such as **el** (clear to end of line sequence) are listed by a two- to five-character capname, an '=', and a string ended by the next occurrence of a comma. A delay in milliseconds may appear anywhere in such a capability, preceded by **$** and enclosed in angle brackets, as in **el=\EK$<3>**. Padding characters are supplied by **tput**. The delay can be any of the following: a number, a number followed by an asterisk, such as **5***, a number followed by a slash, such as **5/**, or a number followed by both, such as **5*/**. A '*****' shows that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. (In the case of insert characters, the factor is still the number of lines affected. This is always 1 unless the device has **in** and the software uses it.) When a '*****' is specified, it is sometimes useful to give a delay of the form **3.5** to specify a delay per unit to tenths of milliseconds. (Only one decimal place is allowed.)

A '/' indicates that the padding is mandatory. If a device has **xon** defined, the padding information is advisory and will only be used for cost estimates or when the device is in raw mode. Mandatory padding will be transmitted regardless of the setting of **xon**. If padding (whether advisory or mandatory) is specified for **bel** or **flash**, however, it will always be used, regardless of whether **xon** is specified.

terminfo offers notation for encoding special characters. Both **\E** and **\e** map to an ESCAPE character, *^x* maps to a control *x* for any appropriate *x*, and the sequences **\n, \l, \r, \t, \b, \f**, and **\s** give a newline, linefeed, return, tab, backspace, formfeed, and space, respectively. Other escapes include: **\^** for caret (^); **** for backslash (\); **\,** for comma (,); **\:** for colon (:); and **\0** for null. (**\0** will actually produce **\200**, which does not terminate a string but behaves as a null character on most devices, providing CS7 is specified. [See **stty**(1).] Finally, characters may be given as three octal digits after a backslash (for example, \123).

Sometimes individual capabilities must be commented out. To do this, put a period before the capability name. For example, see the second **ind** in the example above. Note that capabilities are defined in a left-to-right order and, therefore, a prior definition will override a later definition.

Preparing Descriptions

The most effective way to prepare a device description is by imitating the description of a similar device in **terminfo** and building up a description gradually, using partial descriptions with **vi** to check that they are correct. Be aware that a very unusual device may expose deficiencies in the ability of the **terminfo** file to describe it or the inability of **vi** to work with that device. To test a new device description, set the environment variable **TERMINFO** to the pathname of a directory containing the compiled description you are working on and programs will look there rather than in **/usr/share/lib/terminfo**. To get the padding for insert-line correct (if the device manufacturer did not document it) a severe test is to comment out **xon**, edit a large file at 9600 baud with **vi**, delete 16 or so lines from the middle of the screen, and then press the **u** key several times quickly. If the display is corrupted, more padding is usually needed. A similar test can be used for insert-character.

Section 1-1: Basic Capabilities

The number of columns on each line for the device is given by the **cols** numeric capability. If the device has a screen, then the number of lines on the screen is given by the **lines** capability. If the device wraps around to the beginning of the next line when it reaches the right margin, then it should have the **am** capability. If the terminal can clear its screen, leaving the cursor in the home position, then this is given by the **clear** string capability. If the terminal overstrikes (rather than clearing a position when a character is struck over) then it should have the **os** capability. If the device is a printing terminal, with no soft copy unit, specify both **hc** and **os**. If there is a way to move the cursor to the left edge of the current row, specify this as **cr**. (Normally this will be carriage return, control M.) If there is a way to produce an audible signal (such as a bell or a beep), specify it as **bel**. If, like most devices, the device uses the xon-xoff flow-control protocol, specify **xon**.

If there is a way to move the cursor one position to the left (such as backspace), that capability should be given as **cub1**. Similarly, sequences to move to the right, up, and down should be given as **cuf1**, **cuu1**, and **cud1**, respectively. These local cursor motions must not alter the text they pass over; for example, you would not normally use "**cuf1**=\s" because the space would erase the character moved over.

A very important point here is that the local cursor motions encoded in **terminfo** are undefined at the left and top edges of a screen terminal. Programs should never attempt to backspace around the left edge, unless **bw** is specified, and should never attempt to go up locally off the top. To scroll text up, a program goes to the bottom left corner of the screen and sends the **ind** (index) string.

To scroll text down, a program goes to the top left corner of the screen and sends the **ri** (reverse index) string. The strings **ind** and **ri** are undefined when not on their respective corners of the screen.

Parameterized versions of the scrolling sequences are **indn** and **rin**. These versions have the same semantics as **ind** and **ri**, except that they take one parameter and scroll the number of lines specified by that parameter. They are also undefined except at the appropriate edge of the screen.

The **am** capability tells whether the cursor sticks at the right edge of the screen when text is output, but this does not necessarily apply to a **cuf1** from the last column. Backward motion from the left edge of the screen is possible only when **bw** is specified. In this case, **cub1** will move to the right edge of the previous row. If **bw** is not given, the effect is undefined. This is useful for drawing a box around the edge of the screen, for example. If the device has switch selectable automatic margins, **am** should be specified in the **terminfo** source file. In this case, initialization strings should turn on this option, if possible. If the device has a command that moves to the first column of the next line, that command can be given as **nel** (newline). It does not matter if the command clears the remainder of the current line, so if the device has no **cr** and **lf** it may still be possible to craft a working **nel** out of one or both of them.

These capabilities suffice to describe hardcopy and screen terminals. Thus the AT&T 5320 hardcopy terminal is described as follows:

```
5320|att5320|AT&T 5320 hardcopy terminal,
    am, hc, os,
    cols#132,
    bel=^G, cr=\r, cub1=\b, cnd1=\n,
    dch1=\E[P, dl1=\E[M,
    ind=\n,
```

while the Lear Siegler ADM–3 is described as

```
adm3 | lsi adm3,
  am, bel=^G, clear=^Z, cols#80, cr=^M, cub1=^H,
  cud1=^J, ind=^J, lines#24,
```

Section 1-2: Parameterized Strings

Cursor addressing and other strings requiring parameters are described by a parameterized string capability, with **printf**-like escapes (%*x*) in it. For example, to address the cursor, the **cup** capability is given, using two parameters: the row and column to address to. (Rows and columns are numbered from zero and refer to the physical screen visible to the user, not to any unseen memory.) If the terminal has memory relative cursor addressing, that can be indicated by **mrcup**.

The parameter mechanism uses a stack and special `%` codes to manipulate the stack in the manner of Reverse Polish Notation (postfix). Typically a sequence will push one of the parameters onto the stack and then print it in some format. Often more complex operations are necessary. Operations are in postfix form with the operands in the usual order. That is, to subtract 5 from the first parameter, one would use `%p1%{5}%-`.

The `%` encodings have the following meanings:

`%%` outputs '%'

`%[[:]`*flags*][*width*[.*precision*]][**doxXs**]
: as in **printf**, flags are `[-+#]` and space

`%c` print pop gives %c

`%p[1-9]`
: push *i*th parm

`%P[a-z]`
: set dynamic variable [a-z] to pop

`%g[a-z]`
: get dynamic variable [a-z] and push it

`%P[A-Z]`
: set static variable [a-z] to pop

`%g[A-Z]`
: get static variable [a-z] and push it

`%'`*c*`'` push char constant *c*

`%{`*nn*`}` push decimal constant *nn*

`%l` push strlen(pop)

`%+ %- %* %/ %m`
: arithmetic (`%m` is mod): push(pop $integer_2$ op pop $integer_1$) where $integer_1$ is the top of the stack

`%& %| %^`
: bit operations: push(pop $integer_2$ op pop $integer_1$)

`%= %> %<`
: logical operations: push(pop $integer_2$ op pop $integer_1$)

`%A %O` logical operations: and, or

`%! %~` unary operations: push(op pop)

`%i` (for ANSI terminals) add 1 to first parm, if one parm present, or first two parms, if more than one parm present

`%?` *expr* `%t` *thenpart* `%e` *elsepart* `%;`
: if-then-else, `%e` *elsepart* is optional; else-if's are possible ala Algol 68: `%?` c_1 `%t` b_1 `%e` c_2 `%t` b_2 `%e` c_3 `%t` b_3 `%e` c_4 `%t` b_4 `%e` b_5`%;`
: c_i are conditions, b_i are bodies.

If the "-" flag is used with "%[doxXs]", then a colon (:) must be placed between the "%" and the "-" to differentiate the flag from the binary "%-" operator, for example, "%:-16.16s".

Consider the Hewlett-Packard 2645, which, to get to row 3 and column 12, needs to be sent **\E&a12c03Y** padded for 6 milliseconds. Note that the order of the rows and columns is inverted here, and that the row and column are zero-padded as two digits. Thus its **cup** capability is:

```
        cup=\E&a%p2%2.2dc%p1%2.2dY$<6>
```

The Micro-Term ACT-IV needs the current row and column sent preceded by a **^T**, with the row and column simply encoded in binary, "**cup=^T%p1%c%p2%c**". Devices that use "**%c**" need to be able to backspace the cursor (**cub1**), and to move the cursor up one line on the screen (**cuu1**). This is necessary because it is not always safe to transmit **\n**, **^D**, and **\r**, as the system may change or discard them. (The library routines dealing with **terminfo** set tty modes so that tabs are never expanded, so **\t** is safe to send. This turns out to be essential for the Ann Arbor 4080.)

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus "**cup=\E=%p1%'\s'%+%c%p2%'\s'%+%c**". After sending "**\E=**", this pushes the first parameter, pushes the ASCII value for a space (32), adds them (pushing the sum on the stack in place of the two previous values), and outputs that value as a character. Then the same is done for the second parameter. More c-omplex arithmetic is possible using the stack.

Section 1-3: Cursor Motions

If the terminal has a fast way to home the cursor (to very upper left corner of screen) then this can be given as **home**; similarly a fast way of getting to the lower left-hand corner can be given as **ll**; this may involve going up with **cuu1** from the home position, but a program should never do this itself (unless **ll** does) because it can make no assumption about the effect of moving up from the home position. Note that the home position is the same as addressing to (0,0): to the top left corner of the screen, not of memory. (Thus, the **\EH** sequence on Hewlett-Packard terminals cannot be used for **home** without losing some of the other features on the terminal.)

If the device has row or column absolute-cursor addressing, these can be given as single parameter capabilities **hpa** (horizontal position absolute) and **vpa** (vertical position absolute). Sometimes these are shorter than the more general two-parameter sequence (as with the Hewlett-Packard 2645) and can be used in preference to **cup**. If there are parameterized local motions (for example, move *n* spaces to the right) these can be given as **cud**, **cub**, **cuf**, and **cuu** with a single parameter indicating how many spaces to move. These are primarily useful if the device does not have **cup**, such as the Tektronix 4025.

If the device needs to be in a special mode when running a program that uses these capabilities, the codes to enter and exit this mode can be given as **smcup** and **rmcup**. This arises, for example, from terminals, such as the Concept, with more than one page of memory. If the device has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the device for cursor addressing to work properly. This is also used for the Tektronix

4025, where **smcup** sets the command character to be the one used by **terminfo**. If the **smcup** sequence will not restore the screen after an **rmcup** sequence is output (to the state prior to outputting **rmcup**), specify **nrrmc**.

Section 1-4: Area Clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as **el**. If the terminal can clear from the beginning of the line to the current position inclusive, leaving the cursor where it is, this should be given as **el1**. If the terminal can clear from the current position to the end of the display, then this should be given as **ed**. **ed** is only defined from the first column of a line. (Thus, it can be simulated by a request to delete a large number of lines, if a true **ed** is not available.)

Section 1-5: Insert/Delete Line

If the terminal can open a new blank line before the line where the cursor is, this should be given as **il1**; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line which the cursor is on, then this should be given as **dl1**; this is done only from the first position on the line to be deleted. Versions of **il1** and **dl1** which take a single parameter and insert or delete that many lines can be given as **il** and **dl**.

If the terminal has a settable destructive scrolling region (like the VT100) the command to set this can be described with the **csr** capability, which takes two parameters: the top and bottom lines of the scrolling region. The cursor position is, alas, undefined after using this command. It is possible to get the effect of insert or delete line using this command — the **sc** and **rc** (save and restore cursor) commands are also useful. Inserting lines at the top or bottom of the screen can also be done using **ri** or **ind** on many terminals without a true insert/delete line, and is often faster even on terminals with those features.

To determine whether a terminal has destructive scrolling regions or non-destructive scrolling regions, create a scrolling region in the middle of the screen, place data on the bottom line of the scrolling region, move the cursor to the top line of the scrolling region, and do a reverse index (**ri**) followed by a delete line (**dl1**) or index (**ind**). If the data that was originally on the bottom line of the scrolling region was restored into the scrolling region by the **dl1** or **ind**, then the terminal has non-destructive scrolling regions. Otherwise, it has destructive scrolling regions. Do not specify **csr** if the terminal has non-destructive scrolling regions, unless **ind**, **ri**, **indn**, **rin**, **dl**, and **dl1** all simulate destructive scrolling.

If the terminal has the ability to define a window as part of memory, which all commands affect, it should be given as the parameterized string **wind**. The four parameters are the starting and ending lines in memory and the starting and ending columns in memory, in that order.

If the terminal can retain display memory above, then the **da** capability should be given; if display memory can be retained below, then **db** should be given. These indicate that deleting a line or scrolling a full screen may bring non-blank lines up from below or that scrolling back with **ri** may bring down non-blank lines.

Section 1-6: Insert/Delete Character

There are two basic kinds of intelligent terminals with respect to insert/delete character operations which can be described using `terminfo`. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks. You can determine the kind of terminal you have by clearing the screen and then typing text separated by cursor motions. Type "`abc  def`" using local cursor motions (not spaces) between the `abc` and the `def`. Then position the cursor before the `abc` and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, then your terminal does not distinguish between blanks and untyped positions. If the `abc` shifts over to the `def` which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal, and should give the capability `in`, which stands for "insert null." While these are two logically separate attributes (one line versus multiline insert mode, and special treatment of untyped spaces) we have seen no terminals whose insert mode cannot be described with the single attribute.

`terminfo` can describe both terminals that have an insert mode and terminals which send a simple sequence to open a blank position on the current line. Give as `smir` the sequence to get into insert mode. Give as `rmir` the sequence to leave insert mode. Now give as `ich1` any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give `ich1`; terminals that send a sequence to open a screen position should give it here. (If your terminal has both, insert mode is usually preferable to `ich1`. Do not give both unless the terminal actually requires both to be used in combination.) If post-insert padding is needed, give this as a number of milliseconds padding in `ip` (a string option). Any other sequence which may need to be sent after an insert of a single character may also be given in `ip`. If your terminal needs both to be placed into an 'insert mode' and a special code to precede each inserted character, then both `smir`/rmir and `ich1` can be given, and both will be used. The `ich` capability, with one parameter, *n*, will insert *n* blanks.

If padding is necessary between characters typed while not in insert mode, give this as a number of milliseconds padding in `rmp`.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (for example, if there is a tab after the insertion position). If your terminal allows motion while in insert mode you can give the capability `mir` to speed up inserting in this case. Omitting `mir` will affect only speed. Some terminals (notably Datamedia's) must not have `mir` because of the way their insert mode works.

Finally, you can specify `dch1` to delete a single character, `dch` with one parameter, *n*, to delete *n* characters, and delete mode by giving `smdc` and `rmdc` to enter and exit delete mode (any mode the terminal needs to be placed in for `dch1` to work).

A command to erase *n* characters (equivalent to outputting *n* blanks without moving the cursor) can be given as **ech** with one parameter.

Section 1-7: Highlighting, Underlining, and Visible Bells

Your device may have one or more kinds of display attributes that allow you to highlight selected characters when they appear on the screen. The following display modes (shown with the names by which they are set) may be available: a blinking screen (**blink**), bold or extra-bright characters (**bold**), dim or half-bright characters (**dim**), blanking or invisible text (**invis**), protected text (**prot**), a reverse-video screen (**rev**), and an alternate character set (**smacs** to enter this mode and **rmacs** to exit it). (If a command is necessary before you can enter alternate character set mode, give the sequence in **enacs** or "enable alternate-character-set" mode.) Turning on any of these modes singly may or may not turn off other modes.

sgr0 should be used to turn off all video enhancement capabilities. It should always be specified because it represents the only way to turn off some capabilities, such as **dim** or **blink**.

You should choose one display method as *standout mode* [see **curses**(3curses)] and use it to highlight error messages and other kinds of text to which you want to draw attention. Choose a form of display that provides strong contrast but that is easy on the eyes. (We recommend reverse-video plus half-bright or reverse-video alone.) The sequences to enter and exit standout mode are given as **smso** and **rmso**, respectively. If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, as the TVI 912 and Teleray 1061 do, then **xmc** should be given to tell how many spaces are left.

Sequences to begin underlining and end underlining can be specified as **smul** and **rmul** , respectively. If the device has a sequence to underline the current character and to move the cursor one space to the right (such as the Micro-Term MIME), this sequence can be specified as **uc**.

Terminals with the "magic cookie" glitch (**xmc**) deposit special "cookies" when they receive mode-setting sequences, which affect the display algorithm rather than having extra bits for each character. Some terminals, such as the Hewlett-Packard 2621, automatically leave standout mode when they move to a new line or the cursor is addressed. Programs using standout mode should exit standout mode before moving the cursor or sending a newline, unless the **msgr** capability, asserting that it is safe to move in standout mode, is present.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement), then this can be given as **flash**; it must not move the cursor. A good flash can be done by changing the screen into reverse video, pad for 200 ms, then return the screen to normal video.

If the cursor needs to be made more visible than normal when it is not on the bottom line (to make, for example, a non-blinking underline into an easier to find block or blinking underline) give this sequence as **cvvis**. The boolean **chts** should also be given. If there is a way to make the cursor completely invisible, give that as **civis**. The capability **cnorm** should be given which undoes the effects of either of these modes.

If your terminal generates underlined characters by using the underline character (with no special sequences needed) even though it does not otherwise overstrike characters, then you should specify the capability **ul**. For devices on which a character overstriking another leaves both characters on the screen, specify the capability **os**. If overstrikes are erasable with a blank, then this should be indicated by specifying **eo**.

If there is a sequence to set arbitrary combinations of modes, this should be given as **sgr** (set attributes), taking nine parameters. Each parameter is either **0** or non-zero, as the corresponding attribute is on or off. The nine parameters are, in order: standout, underline, reverse, blink, dim, bold, blank, protect, alternate character set. Not all modes need to be supported by **sgr**; only those for which corresponding separate attribute commands exist should be supported. For example, let's assume that the terminal in question needs the following escape sequences to turn on various modes.

tparm parameter	attribute	escape sequence
	none	`\E[0m`
p1	standout	`\E[0;4;7m`
p2	underline	`\E[0;3m`
p3	reverse	`\E[0;4m`
p4	blink	`\E[0;5m`
p5	dim	`\E[0;7m`
p6	bold	`\E[0;3;4m`
p7	invis	`\E[0;8m`
p8	protect	not available
p9	altcharset	`^O` (off) `^N` (on)

Note that each escape sequence requires a **0** to turn off other modes before turning on its own mode. Also note that, as suggested above, *standout* is set up to be the combination of *reverse* and *dim*. Also, because this terminal has no *bold* mode, *bold* is set up as the combination of *reverse* and *underline*. In addition, to allow combinations, such as *underline+blink*, the sequence to use would be **\E[0;3;5m**. The terminal doesn't have *protect* mode, either, but that cannot be simulated in any way, so **p8** is ignored. The *altcharset* mode is different in that it is either **^O** or **^N**, depending on whether it is off or on. If all modes were to be turned on, the sequence would be **\E[0;3;4;5;7;8m^N**.

Now look at when different sequences are output. For example, **;3** is output when either **p2** or **p6** is true, that is, if either *underline* or *bold* modes are turned on. Writing out the above sequences, along with their dependencies, gives the following:

sequence	when to output	**terminfo** translation
`\E[0`	always	`\E[0`
`;3`	if **p2** or **p6**	`%?%p2%p6%|%t;3%;`
`;4`	if **p1** or **p3** or **p6**	`%?%p1%p3%|%p6%|%t;4%;`
`;5`	if **p4**	`%?%p4%t;5%;`
`;7`	if **p1** or **p5**	`%?%p1%p5%|%t;7%;`

`;8`	if `p7`	`%?%p7%t;8%;`
`m`	always	`m`
`^N` or `^O`	if `p9` `^N`, else `^O`	`%?%p9%t^N%e^O%;`

Putting this all together into the **sgr** sequence gives:

```
sgr=\E[0%?%p2%p6%|%t;3%;%?%p1%p3%|%p6%
        |%t;4%;%?%p5%t;5%;%?%p1%p5%
        |%t;7%;%?%p7%t;8%;m%?%p9%t^N%e^O%;,
```

Remember that **sgr** and **sgr0** must always be specified.

Section 1-8: Keypad

If the device has a keypad that transmits sequences when the keys are pressed, this information can also be specified. Note that it is not possible to handle devices where the keypad only works in local (this applies, for example, to the unshifted Hewlett-Packard 2621 keys). If the keypad can be set to transmit or not transmit, specify these sequences as **smkx** and **rmkx**. Otherwise the keypad is assumed to always transmit.

The sequences sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kcub1, kcuf1, kcuu1, kcud1,** and **khome**, respectively. If there are function keys such as f0, f1, ..., f63, the sequences they send can be specified as **kf0, kf1, ..., kf63**. If the first 11 keys have labels other than the default f0 through f10, the labels can be given as **lf0, lf1, ..., lf10**. The codes transmitted by certain other special keys can be given: **kll** (home down), **kbs** (backspace), **ktbc** (clear all tabs), **kctab** (clear the tab stop in this column), **kclr** (clear screen or erase key), **kdch1** (delete character), **kdl1** (delete line), **krmir** (exit insert mode), **kel** (clear to end of line), **ked** (clear to end of screen), **kich1** (insert character or enter insert mode), **kil1** (insert line), **knp** (next page), **kpp** (previous page), **kind** (scroll forward/down), **kri** (scroll backward/up), **khts** (set a tab stop in this column). In addition, if the keypad has a 3 by 3 array of keys including the four arrow keys, the other five keys can be given as **ka1**, **ka3**, **kb2**, **kc1**, and **kc3**. These keys are useful when the effects of a 3 by 3 directional pad are needed. Further keys are defined above in the capabilities list.

Strings to program function keys can be specified as **pfkey**, **pfloc**, and **pfx**. A string to program screen labels should be specified as **pln**. Each of these strings takes two parameters: a function key identifier and a string to program it with. **pfkey** causes pressing the given key to be the same as the user typing the given string; **pfloc** causes the string to be executed by the terminal in local mode; and **pfx** causes the string to be transmitted to the computer. The capabilities **nlab**, **lw** and **lh** define the number of programmable screen labels and their width and height. If there are commands to turn the labels on and off, give them in **smln** and **rmln**. **smln** is normally output after one or more **pln** sequences to make sure that the change becomes visible.

Section 1-9: Tabs and Initialization

If the device has hardware tabs, the command to advance to the next tab stop can be given as **ht** (usually control I). A "backtab" command that moves leftward to the next tab stop can be given as **cbt**. By convention, if tty modes show that tabs are being expanded by the computer rather than being sent to the device, programs should not use **ht** or **cbt** (even if they are present) because the user may not have the tab stops properly set. If the device has hardware tabs that are initially set every *n* spaces when the device is powered up, the numeric parameter **it** is given, showing the number of spaces the tabs are set to. This is normally used by **tput init** [see **tput**(1)] to determine whether to set the mode for hardware tab expansion and whether to set the tab stops. If the device has tab stops that can be saved in nonvolatile memory, the **terminfo** description can assume that they are properly set. If there are commands to set and clear tab stops, they can be given as **tbc** (clear all tab stops) and **hts** (set a tab stop in the current column of every row).

Other capabilities include: **is1**, **is2**, and **is3**, initialization strings for the device; **iprog**, the path name of a program to be run to initialize the device; and **if**, the name of a file containing long initialization strings. These strings are expected to set the device into modes consistent with the rest of the **terminfo** description. They must be sent to the device each time the user logs in and be output in the following order: run the program **iprog**; output **is1**; output **is2**; set the margins using **mgc**, **smgl** and **smgr**; set the tabs using **tbc** and **hts**; print the file **if**; and finally output **is3**. This is usually done using the **init** option of **tput**.

Most initialization is done with **is2**. Special device modes can be set up without duplicating strings by putting the common sequences in **is2** and special cases in **is1** and **is3**. Sequences that do a reset from a totally unknown state can be given as **rs1**, **rs2**, **rf**, and **rs3**, analogous to **is1**, **is2**, **is3**, and **if**. (The method using files, **if** and **rf**, is used for a few terminals, from **/usr/share/lib/tabset/***; however, the recommended method is to use the initialization and reset strings.) These strings are output by **tput** reset, which is used when the terminal gets into a wedged state. Commands are normally placed in **rs1**, **rs2**, **rs3**, and **rf** only if they produce annoying effects on the screen and are not necessary when logging in. For example, the command to set a terminal into 80-column mode would normally be part of **is2**, but on some terminals it causes an annoying glitch on the screen and is not normally needed because the terminal is usually already in 80-column mode.

If a more complex sequence is needed to set the tabs than can be described by using **tbc** and **hts**, the sequence can be placed in **is2** or **if**.

Any margin can be cleared with **mgc**. (For instructions on how to specify commands to set and clear margins, see "Margins" below under "PRINTER CAPABILITIES.")

Section 1-10: Delays

Certain capabilities control padding in the **tty** driver. These are primarily needed by hard-copy terminals, and are used by **tput init** to set tty modes appropriately. Delays embedded in the capabilities **cr**, **ind**, **cub1**, **ff**, and **tab** can be used to set the appropriate delay bits to be set in the tty driver. If **pb** (padding baud rate) is given, these values can be ignored at baud rates below the value of **pb**.

Section 1-11: Status Lines

If the terminal has an extra "status line" that is not normally used by software, this fact can be indicated. If the status line is viewed as an extra line below the bottom line, into which one can cursor address normally (such as the Heathkit h19's 25th line, or the 24th line of a VT100 which is set to a 23-line scrolling region), the capability **hs** should be given. Special strings that go to a given column of the status line and return from the status line can be given as **tsl** and **fsl**. (**fsl** must leave the cursor position in the same place it was before **tsl**. If necessary, the **sc** and **rc** strings can be included in **tsl** and **fsl** to get this effect.) The capability **tsl** takes one parameter, which is the column number of the status line the cursor is to be moved to.

If escape sequences and other special commands, such as tab, work while in the status line, the flag **eslok** can be given. A string which turns off the status line (or otherwise erases its contents) should be given as **dsl**. If the terminal has commands to save and restore the position of the cursor, give them as **sc** and **rc**. The status line is normally assumed to be the same width as the rest of the screen, for example, **cols**. If the status line is a different width (possibly because the terminal does not allow an entire line to be loaded) the width, in columns, can be indicated with the numeric parameter **wsl**.

Section 1-12: Line Graphics

If the device has a line drawing alternate character set, the mapping of glyph to character would be given in **acsc**. The definition of this string is based on the alternate character set used in the DEC VT100 terminal, extended slightly with some characters from the AT&T 4410v1 terminal.

glyph name	vt100+ character
arrow pointing right	+
arrow pointing left	,
arrow pointing down	.
solid square block	0
lantern symbol	I
arrow pointing up	-
diamond	`
checker board (stipple)	a
degree symbol	f
plus/minus	g
board of squares	h
lower right corner	j
upper right corner	k
upper left corner	l
lower left corner	m
plus	n
scan line 1	o
horizontal line	q
scan line 9	s
left tee (├)	t
right tee (┤)	u
bottom tee (┴)	v
top tee (┬)	w
vertical line	x
bullet	~

The best way to describe a new device's line graphics set is to add a third column to the above table with the characters for the new device that produce the appropriate glyph when the device is in the alternate character set mode. For example,

glyph name	vt100+ char	new tty char
upper left corner	`l`	`R`
lower left corner	`m`	`F`
upper right corner	`k`	`T`
lower right corner	`j`	`G`
horizontal line	`q`	`,`
vertical line	`x`	`.`

Now write down the characters left to right, as in "`acsc=lRmFkTjGq\,x.`".

In addition, **`terminfo`** allows you to define multiple character sets. See Section 2-5 for details.

Section 1-13: Color Manipulation

Let us define two methods of color manipulation: the Tektronix method and the HP method. The Tektronix method uses a set of N predefined colors (usually 8) from which a user can select "current" foreground and background colors. Thus a terminal can support up to N colors mixed into N*N color-pairs to be displayed on the screen at the same time. When using an HP method the user cannot define the foreground independently of the background, or vice-versa. Instead, the user must define an entire color-pair at once. Up to M color-pairs, made from 2*M different colors, can be defined this way. Most existing color terminals belong to one of these two classes of terminals.

The numeric variables **`colors`** and **`pairs`** define the number of colors and color-pairs that can be displayed on the screen at the same time. If a terminal can change the definition of a color (for example, the Tektronix 4100 and 4200 series terminals), this should be specified with **`ccc`** (can change color). To change the definition of a color (Tektronix 4200 method), use **`initc`** (initialize color). It requires four arguments: color number (ranging from 0 to **`colors`**–1) and three RGB (red, green, and blue) values or three HLS colors (Hue, Lightness, Saturation). Ranges of RGB and HLS values are terminal dependent.

Tektronix 4100 series terminals only use HLS color notation. For such terminals (or dual-mode terminals to be operated in HLS mode) one must define a boolean variable **`hls`**; that would instruct the **`curses init_color`** routine to convert its RGB arguments to HLS before sending them to the terminal. The last three arguments to the **`initc`** string would then be HLS values.

If a terminal can change the definitions of colors, but uses a color notation different from RGB and HLS, a mapping to either RGB or HLS must be developed.

To set current foreground or background to a given color, use **`setaf`** (set ANSI foreground) and **`setab`** (set ANSI background). They require one parameter: the number of the color. To initialize a color-pair (HP method), use **`initp`** (initialize pair). It requires seven parameters: the number of a color-pair (range=0 to **`pairs`**–1), and six RGB values: three for the foreground followed by three for the background. (Each of these groups of three should be in the order RGB.) When **`initc`** or **`initp`** are used, RGB or HLS arguments should be in the order "red, green, blue" or "hue, lightness, saturation"), respectively. To make a color-pair current, use **`scp`** (set color-pair). It takes one parameter, the number of a color-pair.

Some terminals (for example, most color terminal emulators for PCs) erase areas of the screen with current background color. In such cases, **bce** (background color erase) should be defined. The variable **op** (original pair) contains a sequence for setting the foreground and the background colors to what they were at the terminal start-up time. Similarly, **oc** (original colors) contains a control sequence for setting all colors (for the Tektronix method) or color-pairs (for the HP method) to the values they had at the terminal start-up time.

Some color terminals substitute color for video attributes. Such video attributes should not be combined with colors. Information about these video attributes should be packed into the **ncv** (no color video) variable. There is a one-to-one correspondence between the nine least significant bits of that variable and the video attributes. The following table depicts this correspondence.

Attribute	Bit Position	Decimal Value
A_STANDOUT	0	1
A_UNDERLINE	1	2
A_REVERSE	2	4
A_BLINK	3	8
A_DIM	4	16
A_BOLD	5	32
A_INVIS	6	64
A_PROTECT	7	128
A_ALTCHARSET	8	256

When a particular video attribute should not be used with colors, the corresponding **ncv** bit should be set to 1; otherwise it should be set to zero. To determine the information to pack into the **ncv** variable, you must add together the decimal values corresponding to those attributes that cannot coexist with colors. For example, if the terminal uses colors to simulate reverse video (bit number 2 and decimal value 4) and bold (bit number 5 and decimal value 32), the resulting value for **ncv** will be 36 (4 + 32).

Section 1-14: Miscellaneous

If the terminal requires other than a null (zero) character as a pad, then this can be given as **pad**. Only the first character of the **pad** string is used. If the terminal does not have a pad character, specify **npc**.

If the terminal can move up or down half a line, this can be indicated with **hu** (half-line up) and **hd** (half-line down). This is primarily useful for superscripts and subscripts on hardcopy terminals. If a hardcopy terminal can eject to the next page (form feed), give this as `ff` (usually control L).

If there is a command to repeat a given character a given number of times (to save time transmitting a large number of identical characters) this can be indicated with the parameterized string **rep**. The first parameter is the character to be repeated and the second is the number of times to repeat it. Thus, **tparm(repeat_char, 'x', 10)** is the same as **xxxxxxxxxx.**

If the terminal has a settable command character, such as the Tektronix 4025, this can be indicated with **cmdch**. A prototype command character is chosen which is used in all capabilities. This character is given in the **cmdch** capability to identify it. The following convention is supported on some UNIX systems: If the environment variable `CC` exists, all occurrences of the prototype character are replaced with the character in `CC`.

Terminal descriptions that do not represent a specific kind of known terminal, such as *switch*, *dialup*, *patch*, and *network*, should include the **gn** (generic) capability so that programs can complain that they do not know how to talk to the terminal. (This capability does not apply to *virtual* terminal descriptions for which the escape sequences are known.) If the terminal is one of those supported by the UNIX system virtual terminal protocol, the terminal number can be given as **vt**. A line-turnaround sequence to be transmitted before doing reads should be specified in **rfi**.

If the device uses xon/xoff handshaking for flow control, give **xon**. Padding information should still be included so that routines can make better decisions about costs, but actual pad characters will not be transmitted. Sequences to turn on and off xon/xoff handshaking may be given in **smxon** and **rmxon**. If the characters used for handshaking are not `^S` and `^Q`, they may be specified with **xonc** and **xoffc**.

If the terminal has a "meta key" which acts as a shift key, setting the 8th bit of any character transmitted, this fact can be indicated with **km**. Otherwise, software will assume that the 8th bit is parity and it will usually be cleared. If strings exist to turn this "meta mode" on and off, they can be given as **smm** and **rmm**.

If the terminal has more lines of memory than will fit on the screen at once, the number of lines of memory can be indicated with **lm**. A value of **lm**#0 indicates that the number of lines is not fixed, but that there is still more memory than fits on the screen.

Media copy strings which control an auxiliary printer connected to the terminal can be given as **mc0**: print the contents of the screen, **mc4**: turn off the printer, and **mc5**: turn on the printer. When the printer is on, all text sent to the terminal will be sent to the printer. A variation, **mc5p**, takes one parameter, and leaves the printer on for as many characters as the value of the parameter, then turns the printer off. The parameter should not exceed 255. If the text is not displayed on the terminal screen when the printer is on, specify **mc5i** (silent printer). All text, including **mc4**, is transparently passed to the printer while an **mc5p** is in effect.

Section 1-15: Special Cases

The working model used by **terminfo** fits most terminals reasonably well. However, some terminals do not completely match that model, requiring special support by **terminfo**. These are not meant to be construed as deficiencies in the terminals; they are just differences between the working model and the actual hardware. They may be unusual devices or, for some reason, do not have all the features of the **terminfo** model implemented.

Terminals that cannot display tilde (~) characters, such as certain Hazeltine terminals, should indicate **hz**.

Terminals that ignore a linefeed immediately after an **am** wrap, such as the Concept 100, should indicate **xenl**. Those terminals whose cursor remains on the right-most column until another character has been received, rather than wrapping immediately upon receiving the right-most character, such as the VT100, should also indicate **xenl**.

If **el** is required to get rid of standout (instead of writing normal text on top of it), **xhp** should be given.

Those Teleray terminals whose tabs turn all characters moved over to blanks, should indicate **xt** (destructive tabs). This capability is also taken to mean that it is not possible to position the cursor on top of a "magic cookie." Therefore, to erase standout mode, it is necessary, instead, to use delete and insert line.

Those Beehive Superbee terminals which do not transmit the escape or control–C characters, should specify **xsb**, indicating that the f1 key is to be used for escape and the f2 key for control C.

Section 1-16: Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability **use** can be given with the name of the similar terminal. The capabilities given before **use** override those in the terminal type invoked by **use**. A capability can be canceled by placing *xx***@** to the left of the capability definition, where *xx* is the capability. For example, the entry

```
att4424-2|Teletype 4424 in display function group ii,
    rev@, sgr@, smul@, use=att4424,
```

defines an AT&T 4424 terminal that does not have the **rev**, **sgr**, and **smul** capabilities, and hence cannot do highlighting. This is useful for different modes for a terminal, or for different user preferences. More than one **use** capability may be given.

PART 2: PRINTER CAPABILITIES

The **terminfo** database allows you to define capabilities of printers as well as terminals. To find out what capabilities are available for printers as well as for terminals, see the two lists under "DEVICE CAPABILITIES" that list capabilities by variable and by capability name.

Section 2-1: Rounding Values

Because parameterized string capabilities work only with integer values, we recommend that **terminfo** designers create strings that expect numeric values that have been rounded. Application designers should note this and should always round values to the nearest integer before using them with a parameterized string capability.

Section 2-2: Printer Resolution

A printer's resolution is defined to be the smallest spacing of characters it can achieve. In general printers have independent resolution horizontally and vertically. Thus the vertical resolution of a printer can be determined by measuring the smallest achievable distance between consecutive printing baselines, while the horizontal resolution can be determined by measuring the smallest achievable distance between the left-most edges of consecutive printed, identical, characters.

All printers are assumed to be capable of printing with a uniform horizontal and vertical resolution. The view of printing that **terminfo** currently presents is one of printing inside a uniform matrix: All characters are printed at fixed positions relative to each "cell" in the matrix; furthermore, each cell has the same size given by the smallest horizontal and vertical step sizes dictated by the resolution. (The cell size can be changed as will be seen later.)

Many printers are capable of "proportional printing," where the horizontal spacing depends on the size of the character last printed. **terminfo** does not make use of this capability, although it does provide enough capability definitions to allow an application to simulate proportional printing.

A printer must not only be able to print characters as close together as the horizontal and vertical resolutions suggest, but also of "moving" to a position an integral multiple of the smallest distance away from a previous position. Thus printed characters can be spaced apart a distance that is an integral multiple of the smallest distance, up to the length or width of a single page.

Some printers can have different resolutions depending on different "modes." In "normal mode," the existing **terminfo** capabilities are assumed to work on columns and lines, just like a video terminal. Thus the old **lines** capability would give the length of a page in lines, and the **cols** capability would give the width of a page in columns. In "micro mode," many **terminfo** capabilities work on increments of lines and columns. With some printers the micro mode may be concomitant with normal mode, so that all the capabilities work at the same time.

Section 2-3: Specifying Printer Resolution

The printing resolution of a printer is given in several ways. Each specifies the resolution as the number of smallest steps per distance:

Specification of Printer Resolution

Characteristic	Number of Smallest Steps
orhi	Steps per inch horizontally
orvi	Steps per inch vertically
orc	Steps per column
orl	Steps per line

When printing in normal mode, each character printed causes movement to the next column, except in special cases described later; the distance moved is the same as the per-column resolution. Some printers cause an automatic movement to the next line when a character is printed in the rightmost position; the distance moved vertically is the same as the per-line resolution. When printing in micro mode, these distances can be different, and may be zero for some printers.

Specification of Printer Resolution

Automatic Motion after Printing	
Normal Mode:	
orc	Steps moved horizontally
orl	Steps moved vertically

Micro Mode:	
mcs	Steps moved horizontally
mls	Steps moved vertically

Some printers are capable of printing wide characters. The distance moved when a wide character is printed in normal mode may be different from when a regular width character is printed. The distance moved when a wide character is printed in micro mode may also be different from when a regular character is printed in micro mode, but the differences are assumed to be related: If the distance moved for a regular character is the same whether in normal mode or micro mode (**mcs**=**orc**), then the distance moved for a wide character is also the same whether in normal mode or micro mode. This doesn't mean the normal character distance is necessarily the same as the wide character distance, just that the distances don't change with a change in normal to micro mode. However, if the distance moved for a regular character is different in micro mode from the distance moved in normal mode (**mcs**<**orc**), the micro mode distance is assumed to be the same for a wide character printed in micro mode, as the table below shows.

Specification of Printer Resolution
Automatic Motion after Printing Wide Character

Normal Mode or Micro Mode (**mcs** = **orc**):	
widcs	Steps moved horizontally
Micro Mode (**mcs** < **orc**):	
mcs	Steps moved horizontally

There may be control sequences to change the number of columns per inch (the character pitch) and to change the number of lines per inch (the line pitch). If these are used, the resolution of the printer changes, but the type of change depends on the printer:

Specification of Printer Resolution
Changing the Character/Line Pitches

cpi	Change character pitch
cpix	If set, **cpi** changes **orhi**, otherwise changes **orc**
lpi	Change line pitch
lpix	If set, **lpi** changes **orvi**, otherwise changes **orl**
chr	Change steps per column
cvr	Change steps per line

The **cpi** and **lpi** string capabilities are each used with a single argument, the pitch in columns (or characters) and lines per inch, respectively. The **chr** and **cvr** string capabilities are each used with a single argument, the number of steps per column and line, respectively.

Using any of the control sequences in these strings will imply a change in some of the values of **orc**, **orhi**, **orl**, and **orvi**. Also, the distance moved when a wide character is printed, **widcs**, changes in relation to **orc**. The distance moved when a character is printed in micro mode, **mcs**, changes similarly, with one exception: if the distance is 0 or 1, then no change is assumed (see items marked with † in the following table).

Programs that use **cpi**, **lpi**, **chr**, or **cvr** should recalculate the printer resolution (and should recalculate other values see "Effect of Changing Printing Resolution" under "Dot-Mapped Graphics").

Specification of Printer Resolution
Effects of Changing the Character/Line Pitches

Before	*After*
Using **cpi** with **cpix** clear:	
orhi′	**orhi**
orc′	$\mathbf{orc} = \frac{\mathbf{orhi}}{V_{cpi}}$
Using **cpi** with **cpix** set:	
orhi′	$\mathbf{orhi} = \mathbf{orc} \cdot V_{cpi}$
orc′	**orc**
Using **lpi** with **lpix** clear:	
orvi′	**orvi**
orl′	$\mathbf{orl} = \frac{\mathbf{orvi}}{V_{lpi}}$
Using **lpi** with **lpix** set:	
orvi′	$\mathbf{orvi} = \mathbf{orl} \cdot V_{lpi}$
orl′	**orl**
Using **chr**:	
orhi′	**orhi**
orc′	V_{chr}
Using **cvr**:	
orvi′	**orvi**
orl′	V_{cvr}
Using **cpi** or **chr**:	
widcs′	$\mathbf{widcs} = \mathbf{widcs}' \frac{\mathbf{orc}}{\mathbf{orc}'}$
mcs′	$\mathbf{mcs} = \mathbf{mcs}' \frac{\mathbf{orc}}{\mathbf{orc}'}$

V_{cpi}, V_{lpi}, V_{chr}, and V_{cvr} are the arguments used with **cpi**, **lpi**, **chr**, and **cvr**, respectively. The prime marks (′) indicate the old values.

Section 2-4: Capabilities that Cause Movement

In the following descriptions, "movement" refers to the motion of the "current position." With video terminals this would be the cursor; with some printers this is the carriage position. Other printers have different equivalents. In general, the current position is where a character would be displayed if printed.

`terminfo` has string capabilities for control sequences that cause movement a number of full columns or lines. It also has equivalent string capabilities for control sequences that cause movement a number of smallest steps.

String Capabilities for Motion	
mcub1	Move 1 step left
mcuf1	Move 1 step right
mcuu1	Move 1 step up
mcud1	Move 1 step down
mcub	Move *N* steps left
mcuf	Move *N* steps right
mcuu	Move *N* steps up
mcud	Move *N* steps down
mhpa	Move *N* steps from the left
mvpa	Move *N* steps from the top

The latter six strings are each used with a single argument, *N*.

Sometimes the motion is limited to less than the width or length of a page. Also, some printers don't accept absolute motion to the left of the current position. `terminfo` has capabilities for specifying these limits.

Limits to Motion	
mjump	Limit on use of **mcub1**, **mcuf1**, **mcuu1**, **mcud1**
maddr	Limit on use of **mhpa**, **mvpa**
xhpa	If set, **hpa** and **mhpa** can't move left
xvpa	If set, **vpa** and **mvpa** can't move up

If a printer needs to be in a "micro mode" for the motion capabilities described above to work, there are string capabilities defined to contain the control sequence to enter and exit this mode. A boolean is available for those printers where using a carriage return causes an automatic return to normal mode.

Entering/Exiting Micro Mode	
smicm	Enter micro mode
rmicm	Exit micro mode
crxm	Using **cr** exits micro mode

The movement made when a character is printed in the rightmost position varies among printers. Some make no movement, some move to the beginning of the next line, others move to the beginning of the same line. `terminfo` has boolean capabilities for describing all three cases.

	What Happens After Character Printed in Rightmost Position
sam	Automatic move to beginning of same line

Some printers can be put in a mode where the normal direction of motion is reversed. This mode can be especially useful when there are no capabilities for leftward or upward motion, because those capabilities can be built from the motion reversal capability and the rightward or downward motion capabilities. It is best to leave it up to an application to build the leftward or upward capabilities, though, and not enter them in the **terminfo** database. This allows several reverse motions to be strung together without intervening wasted steps that leave and reenter reverse mode.

	Entering/Exiting Reverse Modes
slm	Reverse sense of horizontal motions
rlm	Restore sense of horizontal motions
sum	Reverse sense of vertical motions
rum	Restore sense of vertical motions
While sense of horizontal motions reversed:	
mcub1	Move 1 step right
mcuf1	Move 1 step left
mcub	Move *N* steps right
mcuf	Move *N* steps left
cub1	Move 1 column right
cuf1	Move 1 column left
cub	Move *N* columns right
cuf	Move *N* columns left
While sense of vertical motions reversed:	
mcuu1	Move 1 step down
mcud1	Move 1 step up
mcuu	Move *N* steps down
mcud	Move *N* steps up
cuu1	Move 1 line down
cud1	Move 1 line up
cuu	Move *N* lines down
cud	Move *N* lines up

The reverse motion modes should not affect the **mvpa** and **mhpa** absolute motion capabilities. The reverse vertical motion mode should, however, also reverse the action of the line "wrapping" that occurs when a character is printed in the right-most position. Thus printers that have the standard **terminfo** capability **am** defined should experience motion to the beginning of the previous line when a character is printed in the right-most position under reverse vertical motion mode.

The action when any other motion capabilities are used in reverse motion modes is not defined; thus, programs must exit reverse motion modes before using other motion capabilities.

Two miscellaneous capabilities complete the list of new motion capabilities. One of these is needed for printers that move the current position to the beginning of a line when certain control characters, such as "line-feed" or "form-feed," are used. The other is used for the capability of suspending the motion that normally occurs after printing a character.

Miscellaneous Motion Strings	
`docr`	List of control characters causing `cr`
`zerom`	Prevent auto motion after printing next single character

Margins

`terminfo` provides two strings for setting margins on terminals: one for the left and one for the right margin. Printers, however, have two additional margins, for the top and bottom margins of each page. Furthermore, some printers require not using motion strings to move the current position to a margin and then fixing the margin there, but require the specification of where a margin should be regardless of the current position. Therefore `terminfo` offers six additional strings for defining margins with printers.

Setting Margins	
`smgl`	Set left margin at current column
`smgr`	Set right margin at current column
`smgb`	Set bottom margin at current line
`smgt`	Set top margin at current line
`smgbp`	Set bottom margin at line *N*
`smglp`	Set left margin at column *N*
`smgrp`	Set right margin at column *N*
`smgtp`	Set top margin at line *N*

The last four strings are used with one or more arguments that give the position of the margin or margins to set. If both of **smglp** and **smgrp** are set, each is used with a single argument, *N*, that gives the column number of the left and right margin, respectively. If both of **smgtp** and **smgbp** are set, each is used to set the top and bottom margin, respectively: **smgtp** is used with a single argument, *N*, the line number of the top margin; however, **smgbp** is used with two arguments, *N* and *M*, that give the line number of the bottom margin, the first counting from the top of the page and the second counting from the bottom. This accommodates the two styles of specifying the bottom margin in different manufacturers' printers. When coding a `terminfo` entry for a printer that has a settable bottom margin, only the first or second parameter should be used, depending on the printer. When writing an application that uses **smgbp** to set the bottom margin, both arguments must be given.

If only one of **smglp** and **smgrp** is set, then it is used with two arguments, the column number of the left and right margins, in that order. Likewise, if only one of **smgtp** and **smgbp** is set, then it is used with two arguments that give the top and bottom margins, in that order, counting from the top of the page. Thus when coding a `terminfo` entry for a printer that requires setting both left and right or top and bottom margins simultaneously, only one of **smglp** and **smgrp** or **smgtp** and **smgbp** should be defined; the other should be left blank. When writing an application that uses these string capabilities, the pairs should be first checked to see if each in the pair is set or only one is set, and should then be used accordingly.

In counting lines or columns, line zero is the top line and column zero is the left-most column. A zero value for the second argument with **smgbp** means the bottom line of the page.

All margins can be cleared with **mgc**.

Shadows, Italics, Wide Characters, Superscripts, Subscripts

Five new sets of strings are used to describe the capabilities printers have of enhancing printed text.

	Enhanced Printing
sshm	Enter shadow-printing mode
rshm	Exit shadow-printing mode
sitm	Enter italicizing mode
ritm	Exit italicizing mode
swidm	Enter wide character mode
rwidm	Exit wide character mode
ssupm	Enter superscript mode
rsupm	Exit superscript mode
supcs	List of characters available as superscripts
ssubm	Enter subscript mode
rsubm	Exit subscript mode
subcs	List of characters available as subscripts

If a printer requires the **sshm** control sequence before every character to be shadow-printed, the **rshm** string is left blank. Thus programs that find a control sequence in **sshm** but none in **rshm** should use the **sshm** control sequence before every character to be shadow-printed; otherwise, the **sshm** control sequence should be used once before the set of characters to be shadow-printed, followed by **rshm**. The same is also true of each of the **sitm**/**ritm**, **swidm**/**rwidm**, **ssupm**/**rsupm**, and **ssubm**/ **rsubm** pairs.

Note that **terminfo** also has a capability for printing emboldened text (**bold**). While shadow printing and emboldened printing are similar in that they "darken" the text, many printers produce these two types of print in slightly different ways. Generally, emboldened printing is done by overstriking the same character one or more times. Shadow printing likewise usually involves overstriking, but with a slight movement up and/or to the side so that the character is "fatter."

It is assumed that enhanced printing modes are independent modes, so that it would be possible, for instance, to shadow print italicized subscripts.

As mentioned earlier, the amount of motion automatically made after printing a wide character should be given in **widcs**.

If only a subset of the printable ASCII characters can be printed as superscripts or subscripts, they should be listed in **supcs** or **subcs** strings, respectively. If the **ssupm** or **ssubm** strings contain control sequences, but the corresponding **supcs** or

subcs strings are empty, it is assumed that all printable ASCII characters are available as superscripts or subscripts.

Automatic motion made after printing a superscript or subscript is assumed to be the same as for regular characters. Thus, for example, printing any of the following three examples will result in equivalent motion:

`Bi` $\mathtt{B}_{\mathtt{i}}$ $\mathtt{B}^{\mathtt{i}}$

Note that the existing **msgr** boolean capability describes whether motion control sequences can be used while in "standout mode." This capability is extended to cover the enhanced printing modes added here. **msgr** should be set for those printers that accept any motion control sequences without affecting shadow, italicized, widened, superscript, or subscript printing. Conversely, if **msgr** is not set, a program should end these modes before attempting any motion.

Section 2-5: Alternate Character Sets

In addition to allowing you to define line graphics (described in Section 1-12), **terminfo** lets you define alternate character sets. The following capabilities cover printers and terminals with multiple selectable or definable character sets.

Alternate Character Sets

scs	Select character set *N*
scsd	Start definition of character set *N*, *M* characters
defc	Define character *A*, *B* dots wide, descender *D*
rcsd	End definition of character set *N*
csnm	List of character set names
daisy	Printer has manually changed print-wheels

The **scs**, **rcsd**, and **csnm** strings are used with a single argument, *N*, a number from 0 to 63 that identifies the character set. The **scsd** string is also used with the argument *N* and another, *M*, that gives the number of characters in the set. The **defc** string is used with three arguments: *A* gives the ASCII code representation for the character, *B* gives the width of the character in dots, and *D* is zero or one depending on whether the character is a "descender" or not. The **defc** string is also followed by a string of "image-data" bytes that describe how the character looks (see below).

Character set 0 is the default character set present after the printer has been initialized. Not every printer has 64 character sets, of course; using **scs** with an argument that doesn't select an available character set should cause a null result from **tparm**.

If a character set has to be defined before it can be used, the **scsd** control sequence is to be used before defining the character set, and the **rcsd** is to be used after. They should also cause a null result from **tparm** when used with an argument *N* that doesn't apply. If a character set still has to be selected after being defined, the **scs** control sequence should follow the **rcsd** control sequence. By examining the results of using each of the **scs**, **scsd**, and **rcsd** strings with a character set number in a call to **tparm**, a program can determine which of the three are needed.

Between use of the **scsd** and **rcsd** strings, the **defc** string should be used to define each character. To print any character on printers covered by **terminfo**, the ASCII code is sent to the printer. This is true for characters in an alternate set as well as "normal" characters. Thus the definition of a character includes the ASCII code that represents it. In addition, the width of the character in dots is given, along with an indication of whether the character should descend below the print line (such as the lower case letter "g" in most character sets). The width of the character in dots also indicates the number of image-data bytes that will follow the **defc** string. These image-data bytes indicate where in a dot-matrix pattern ink should be applied to "draw" the character; the number of these bytes and their form are defined below under "Dot-Mapped Graphics."

It's easiest for the creator of **terminfo** entries to refer to each character set by number; however, these numbers will be meaningless to the application developer. The **csnm** string alleviates this problem by providing names for each number.

When used with a character set number in a call to **tparm**, the **csnm** string will produce the equivalent name. These names should be used as a reference only. No naming convention is implied, although anyone who creates a **terminfo** entry for a printer should use names consistent with the names found in user documents for the printer. Application developers should allow a user to specify a character set by number (leaving it up to the user to examine the **csnm** string to determine the correct number), or by name, where the application examines the **csnm** string to determine the corresponding character set number.

These capabilities are likely to be used only with dot-matrix printers. If they are not available, the strings should not be defined. For printers that have manually changed print-wheels or font cartridges, the boolean **daisy** is set.

Section 2-6: Dot-Matrix Graphics

Dot-matrix printers typically have the capability of reproducing "raster-graphics" images. Three new numeric capabilities and three new string capabilities can help a program draw raster-graphics images independent of the type of dot-matrix printer or the number of pins or dots the printer can handle at one time.

Dot-Matrix Graphics

npins	Number of pins, *N*, in print-head
spinv	Spacing of pins vertically in pins per inch
spinh	Spacing of dots horizontally in dots per inch
porder	Matches software bits to print-head pins
sbim	Start printing bit image graphics, *B* bits wide
rbim	End printing bit image graphics

The **sbim** sring is used with a single argument, *B*, the width of the image in dots.

The model of dot-matrix or raster-graphics that **terminfo** presents is similar to the technique used for most dot-matrix printers: each pass of the printer's print-head is assumed to produce a dot-matrix that is *N* dots high and *B* dots wide. This is typically a wide, squat, rectangle of dots. The height of this rectangle in dots will vary from one printer to the next; this is given in the **npins** numeric capability. The size of the rectangle in fractions of an inch will also vary; it can be deduced from the **spinv** and **spinh** numeric capabilities. With these three values an application can divide a complete raster-graphics image into several horizontal strips, perhaps interpolating to account for different dot spacing vertically and horizontally.

The **sbim** and **rbim** strings are used to start and end a dot-matrix image, respectively. The **sbim** string is used with a single argument that gives the width of the dot-matrix in dots. A sequence of "image-data bytes" are sent to the printer after the **sbim** string and before the **rbim** string. The number of bytes is a integral multiple of the width of the dot-matrix; the multiple and the form of each byte is determined by the **porder** string as described below.

The **porder** string is a comma separated list of pin numbers optionally followed by an numerical offset. The offset, if given, is separated from the list with a semicolon. The position of each pin number in the list corresponds to a bit in an 8-bit data byte. The pins are numbered consecutively from 1 to **npins**, with 1 being the top pin. Note that the term "pin" is used loosely here; "ink-jet" dot-matrix printers don't have pins, but can be considered to have an equivalent method of applying a single dot of ink to paper. The bit positions in **porder** are in groups of 8, with the first position in each group the most significant bit and the last position the least significant bit. An application produces 8-bit bytes in the order of the groups in **porder**.

An application computes the "image-data bytes" from the internal image, mapping vertical dot positions in each print-head pass into 8-bit bytes, using a 1 bit where ink should be applied and 0 where no ink should be applied. This can be reversed (0 bit for ink, 1 bit for no ink) by giving a negative pin number. If a position is skipped in **porder**, a 0 bit is used. If a position has a lower case 'x' instead of a pin number, a 1 bit is used in the skipped position. For consistency, a lower case 'o' can be used to represent a 0 filled, skipped bit. There must be a multiple of 8 bit positions used or skipped in **porder**; if not, 0 bits are used to fill the last byte in the least significant bits. The offset, if given, is added to each data byte; the offset can be negative.

Some examples may help clarify the use of the **porder** string. The AT&T 470, AT&T 475 and C.Itoh 8510 printers provide eight pins for graphics. The pins are identified top to bottom by the 8 bits in a byte, from least significant to most. The **porder** strings for these printers would be **8,7,6,5,4,3,2,1**. The AT&T 478 and AT&T 479 printers also provide eight pins for graphics. However, the pins are identified in the reverse order. The **porder** strings for these printers would be **1,2,3,4,5,6,7,8**. The AT&T 5310, AT&T 5320, DEC LA100, and DEC LN03 printers provide six pins for graphics. The pins are identified top to bottom by the decimal values 1, 2, 4, 8, 16 and 32. These correspond to the low six bits in an 8-bit byte, although the decimal values are further offset by the value 63. The **porder** string for these printers would be **,,6,5,4,3,2,1;63**, or alternately **o,o,6,5,4,3,2,1;63**.

Section 2-7: Effect of Changing Printing Resolution

If the control sequences to change the character pitch or the line pitch are used, the pin or dot spacing may change:

Dot-Matrix Graphics
Changing the Character/Line Pitches

`cpi`	Change character pitch
`cpix`	If set, `cpi` changes **spinh**
`lpi`	Change line pitch
`lpix`	If set, `lpi` changes **spinv**

Programs that use `cpi` or `lpi` should recalculate the dot spacing:

Dot-Matrix Graphics
Effects of Changing the Character/Line Pitches

Before	*After*
Using `cpi` with `cpix` clear:	
spinh′	**spinh**
Using `cpi` with `cpix` set:	
spinh′	$\mathbf{spinh} = \mathbf{spinh}' \cdot \frac{\mathbf{orhi}}{\mathbf{orhi}'}$
Using `lpi` with `lpix` clear:	
spinv′	**spinv**
Using `lpi` with `lpix` set:	
spinv′	$\mathbf{spinv} = \mathbf{spinv}' \cdot \frac{\mathbf{orhi}}{\mathbf{orhi}'}$
Using `chr`:	
spinh′	**spinh**
Using `cvr`:	
spinv′	**spinv**

orhi′ and **orhi** are the values of the horizontal resolution in steps per inch, before using `cpi` and after using `cpi`, respectively. Likewise, **orvi′** and **orvi** are the values of the vertical resolution in steps per inch, before using `lpi` and after using `lpi`, respectively. Thus, the changes in the dots per inch for dot-matrix graphics follow the changes in steps per inch for printer resolution.

Section 2-8: Print Quality

Many dot-matrix printers can alter the dot spacing of printed text to produce near "letter quality" printing or "draft quality" printing. Usually it is important to be able to choose one or the other because the rate of printing generally falls off as the quality improves. There are three new strings used to describe these capabilities.

Print Quality	
snlq	Set near-letter quality print
snrmq	Set normal quality print
sdrfq	Set draft quality print

The capabilities are listed in decreasing levels of quality. If a printer doesn't have all three levels, one or two of the strings should be left blank as appropriate.

Section 2-9: Printing Rate and Buffer Size

Because there is no standard protocol that can be used to keep a program synchronized with a printer, and because modern printers can buffer data before printing it, a program generally cannot determine at any time what has been printed. Two new numeric capabilities can help a program estimate what has been printed.

Print Rate/Buffer Size	
cps	Nominal print rate in characters per second
bufsz	Buffer capacity in characters

cps is the nominal or average rate at which the printer prints characters; if this value is not given, the rate should be estimated at one-tenth the prevailing baud rate. **bufsz** is the maximum number of subsequent characters buffered before the guaranteed printing of an earlier character, assuming proper flow control has been used. If this value is not given it is assumed that the printer does not buffer characters, but prints them as they are received.

As an example, if a printer has a 1000-character buffer, then sending the letter "a" followed by 1000 additional characters is guaranteed to cause the letter "a" to print. If the same printer prints at the rate of 100 characters per second, then it should take 10 seconds to print all the characters in the buffer, less if the buffer is not full. By keeping track of the characters sent to a printer, and knowing the print rate and buffer size, a program can synchronize itself with the printer.

Note that most printer manufacturers advertise the maximum print rate, not the nominal print rate. A good way to get a value to put in for **cps** is to generate a few pages of text, count the number of printable characters, and then see how long it takes to print the text.

Applications that use these values should recognize the variability in the print rate. Straight text, in short lines, with no embedded control sequences will probably print at close to the advertised print rate and probably faster than the rate in **cps**. Graphics data with a lot of control sequences, or very long lines of text, will print at well below the advertised rate and below the rate in **cps**. If the application is using **cps** to decide how long it should take a printer to print a block of text, the application should pad the estimate. If the application is using **cps** to decide how much text has already been printed, it should shrink the estimate. The application will thus err in favor of the user, who wants, above all, to see all the output in its correct place.

FILES

/usr/share/lib/terminfo/?/*	compiled terminal description database
/usr/share/lib/tabset/*	tab settings for some terminals, in a format appropriate to be output to the terminal (escape sequences that set margins and tabs)

SEE ALSO

curses(3curses), **ls**(1), **pg**(1), **printf**(3S), **stty**(1), **tic**(1M), **tput**(1), **tty**(1), **vi**(1)

NOTES

The most effective way to prepare a terminal description is by imitating the description of a similar terminal in **terminfo** and to build up a description gradually, using partial descriptions with a screen oriented editor, such as **vi**, to check that they are correct. To easily test a new terminal description the environment variable **TERMINFO** can be set to the pathname of a directory containing the compiled description, and programs will look there rather than in **/usr/share/lib/terminfo**.

timezone (4)

NAME

`timezone` – set default system time zone

SYNOPSIS

`/etc/TIMEZONE`

DESCRIPTION

This file sets and exports the time zone environmental variable **`TZ`**.

This file is "dotted" into other files that must know the time zone.

EXAMPLES

`/etc/TIMEZONE` for the United States east coast:

```
#    Time Zone
TZ=EST5EDT
export TZ
```

SEE ALSO

`ctime`(3C), **`environ`**(5), **`profile`**(4), **`rc2`**(1M)

NAME

ts_dptbl – time-sharing dispatcher parameter table

DESCRIPTION

The process scheduler (or dispatcher) is the portion of the kernel that controls allocation of the CPU to processes. The scheduler supports the notion of scheduling classes where each class defines a scheduling policy, used to schedule processes within that class. Associated with each scheduling class is a set of priority queues on which ready to run processes are linked. These priority queues are mapped by the system configuration into a set of global scheduling priorities which are available to processes within the class. (The dispatcher always selects for execution the process with the highest global scheduling priority in the system.) The priority queues associated with a given class are viewed by that class as a contiguous set of priority levels numbered from 0 (lowest priority) to *n* (highest priority—a configuration-dependent value). The set of global scheduling priorities that the queues for a given class are mapped into might not start at zero and might not be contiguous (depending on the configuration).

Processes in the time-sharing class which are running in user mode (or in kernel mode before going to sleep) are scheduled according to the parameters in a time-sharing dispatcher parameter table (**ts_dptbl**). (Time-sharing processes running in kernel mode after sleeping are run within a special range of priorities reserved for such processes and are not affected by the parameters in the **ts_dptbl** until they return to user mode.) The **ts_dptbl** consists of an array of parameter structures (**struct ts_dpent**), one for each of the *n* priority levels used by time-sharing processes in user mode. The properties of a given priority level *i* are specified by the *i*th parameter structure in this array (**ts_dptbl***i*).

A parameter structure consists of the following members. These are also described in the **/usr/include/sys/ts.h** header file.

ts_globpri	The global scheduling priority associated with this priority level. The mapping between time-sharing priority levels and global scheduling priorities is determined at boot time by the system configuration. **ts_globpri** is the only member of the **ts_dptbl** which cannot be changed with **dispadmin**(1M).
ts_quantum	The length of the time quantum allocated to processes at this level in ticks (HZ).
ts_tqexp	Priority level of the new queue on which to place a process running at the current level if it exceeds its time quantum. Normally this field links to a lower priority time-sharing level that has a larger quantum.
ts_slpret	Priority level of the new queue on which to place a process, that was previously in user mode at this level, when it returns to user mode after sleeping. Normally this field links to a higher priority level that has a smaller quantum.
ts_maxwait	A per process counter, **ts_dispwait** is initialized to zero each time a time-sharing process is placed back on the dispatcher queue after its time quantum has expired or when it is awakened (**ts_dispwait** is not reset to zero when a process is preempted

by a higher priority process). This counter is incremented once per second for each process on the dispatcher queue. If a process's **ts_dispwait** value exceeds the **ts_maxwait** value for its level, the process's priority is changed to that indicated by **ts_lwait**. The purpose of this field is to prevent starvation.

ts_lwait Move a process to this new priority level if **ts_dispwait** is greater than **ts_maxwait**.

An administrator can affect the behavior of the time-sharing portion of the scheduler by reconfiguring the **ts_dptbl**. There are two methods available for doing this.

DISPADMIN CONFIGURATION FILE

With the exception of **ts_globpri** all of the members of the **ts_dptbl** can be examined and modified on a running system using the **dispadmin**(1M) command. Invoking **dispadmin** for the time-sharing class allows the administrator to retrieve the current **ts_dptbl** configuration from the kernel's in-core table, or overwrite the in-core table with values from a configuration file. The configuration file used for input to **dispadmin** must conform to the specific format described below.

Blank lines are ignored and any part of a line to the right of a # symbol is treated as a comment. The first non-blank, non-comment line must indicate the resolution to be used for interpreting the **ts_quantum** time quantum values. The resolution is specified as

RES=*res*

where *res* is a positive integer between 1 and 1,000,000,000 inclusive and the resolution used is the reciprocal of *res* in seconds (for example, **RES=1000** specifies millisecond resolution). Although very fine (nanosecond) resolution may be specified, the time quantum lengths are rounded up to the next integral multiple of the system clock's resolution. The system clock's resolution is hardware-dependent; this resolution can be calculated from the value of **HZ**, which is defined in the file **/usr/include/sys/param.h**. **HZ** gives the number of clock ticks per second of the system clock. For example, an **HZ** of 100 specifies 100 clock ticks per second, or one tick every 10 milliseconds (that is, this system clock has a resolution of 10 milliseconds). If the **-t** and **-r** options are used to specify a time quantum of 34 milliseconds, it is rounded up to 4 ticks (40 milliseconds) on a machine with an **HZ** of 100.

The remaining lines in the file are used to specify the parameter values for each of the time-sharing priority levels. The first line specifies the parameters for time-sharing level 0, the second line specifies the parameters for time-sharing level 1, etc. There must be exactly one line for each configured time-sharing priority level.

EXAMPLE

The following excerpt from a **dispadmin** configuration file illustrates the format. Note that for each line specifying a set of parameters there is a comment indicating the corresponding priority level. These level numbers indicate priority within the time-sharing class, and the mapping between these time-sharing priorities and the corresponding global scheduling priorities is determined by the configuration specified in the **ts** master file. The level numbers are strictly for the convenience of the administrator reading the file and, as with any comment, they are ignored by

dispadmin. **dispadmin** assumes that the lines in the file are ordered by consecutive, increasing priority level (from 0 to the maximum configured time-sharing priority). The level numbers in the comments should normally agree with this ordering; if for some reason they don't, however, **dispadmin** is unaffected.

```
# Time-Sharing Dispatcher Configuration File
RES=1000

# ts_quantum ts_tqexp ts_slpret ts_maxwait ts_lwait  PRIORITY LEVEL
      500       0        10         5         10          #  0
      500       0        11         5         11          #  1
      500       1        12         5         12          #  2
      500       1        13         5         13          #  3
      500       2        14         5         14          #  4
      500       2        15         5         15          #  5
      450       3        16         5         16          #  6
      450       3        17         5         17          #  7
       .        .         .         .          .          .  .
       .        .         .         .          .          .  .
       .        .         .         .          .          .  .
       50      48        59         5         59          #  58
       50      49        59         5         59          #  59
```

FILES

/usr/include/sys/ts.h

SEE ALSO

dispadmin(1M), **priocntl**(1), **priocntl**(2)

NOTES

dispadmin does some limited sanity checking on the values supplied in the configuration file. The sanity checking is intended to ensure that the new **ts_dptbl** values do not cause the system to panic. The sanity checking does not attempt to analyze the effect that the new values will have on the performance of the system. Unusual **ts_dptbl** configurations may have a dramatic negative impact on the performance of the system.

No sanity checking is done on the **ts_dptbl** values specified in the **ts** master file. Specifying an inconsistent or nonsensical **ts_dptbl** configuration through the **ts** master file could cause serious performance problems and/or cause the system to panic.

NAME

ttydefs – file contains terminal line settings information for **ttymon**

DESCRIPTION

/etc/ttydefs is an administrative file that contains information used by **ttymon** to set up the speed and terminal settings for a TTY port.

The **ttydefs** file contains the following fields:

ttylabel	The string **ttymon** tries to match against the TTY port's *ttylabel* field in the port monitor administrative file. It often describes the speed at which the terminal is supposed to run, for example, **1200**.
initial-flags	Contains the initial **termio**(7) settings to which the terminal is to be set. For example, the system administrator will be able to specify what the default erase and kill characters will be. *initial-flags* must be specified in the syntax recognized by the **stty** command.
final-flags	*final-flags* must be specified in the same format as *initial-flags*. **ttymon** sets these final settings after a connection request has been made and immediately prior to invoking a port's service.
autobaud	If the autobaud field contains the character 'A', autobaud will be enabled. Otherwise, autobaud will be disabled. **ttymon** determines what line speed to set the TTY port to by analyzing the carriage returns entered. If autobaud has been disabled, the hunt sequence is used for baud rate determination.
nextlabel	If the user indicates that the current terminal setting is not appropriate by sending a BREAK, **ttymon** searchs for a **ttydefs** entry whose *ttylabel* field matches the *nextlabel* field. If a match is found, **ttymon** uses that field as its *ttylabel* field. A series of speeds is often linked together in this way into a closed set called a hunt sequence. For example, **4800** may be linked to **1200**, which in turn is linked to **2400**, which is finally linked to **4800**.

SEE ALSO

sttydefs(1M), **ttymon**(1M)

NAME

ttysrch – directory search list for ttymap and ttyname

DESCRIPTION

ttysrch is an optional file used by the ttymap(1M) administrative command. The ttymap command creates a map file, /var/tmp/ttymap, used by ttyname(3C) for fast lookups of terminal device names.

The ttysrch file lists the names of directories in /dev that contain terminal and terminal-related device files, as well as the names of directories that contain no such files. The purpose of this file is to improve the performance of ttyname by identifying subdirectories in /dev to be searched first and subdirectories to be ignored. These subdirectory names must appear on separate lines and must begin with /dev. Those path names that do not begin with /dev are ignored and a warning is sent to the console. Blank lines (lines containing only white space) and lines beginning with the comment character "#" are ignored. ttymap writes entries into the mapfile /var/tmp/ttymap in the order in which they occur in the ttysrch file. Subdirectories to be ignored are also specified as such in the mapfile. With the exception of /dev, entries in the ttysrch file are used recursively to identify a directory sub-tree.

When ttyname searches for device files, it tries to find a file whose major/minor device number, file system identifier, and inode number match those of the file descriptor it was given as an argument. If it does not find a match, it settles for a match of just major/minor device and file system identifier, if one can be found. However, if the file descriptor is associated with a cloned device [see clone(7)], this algorithm does not work efficiently because the inode number of the device file associated with a clonable device never matches the inode number of the file descriptor that was returned by the open of that clonable device. To help with these situations, entries can be put into the /etc/ttysrch file to improve performance when cloned devices are used as terminals on a system (for remote login, for example). However, this is useful only if the minor devices related to a cloned device are put into a subdirectory. (It is important to note that device files need not exist for cloned devices; if they do, ttyname fails.) For example, if /dev/starlan is a cloned device, there could be a subdirectory /dev/slan that contains files 0, 1, 2, . . . that correspond to the minor devices of the starlan driver.

An optional second field is used in the /etc/ttysrch file to indicate the matching criteria. This field is separated from the first field by whitespace (any combination of blanks or tabs). The field is made up of a combination of the following letters:

M	major/minor device number
F	file system identifier
I	inode number
X	ignore this directory completely

If this field is not specified for an entry, the default is MFI, which means try to match on all three. For cloned devices the field should be MF, which indicates that it is not necessary to match on the inode number.

Without the **/etc/ttysrch** file, **ttymap** maps the **/dev** directory by first looking in the directories **/dev/term**, **/dev/pts**, and **/dev/xt**, and by ignoring **/dev/dsk** and **/dev/rdsk**. If a system has terminal devices installed in directories other than these, it may help performance if the **ttysrch** file is created and contains that list of directories.

EXAMPLE

A sample **/etc/ttysrch** file follows:

```
/dev/term      MFI
/dev/pts       MFI
/dev/xt        MFI
/dev/slan      MF
/dev/dsk       X
/dev/rdsk      X
```

This file tells **ttyname** that it should first search through those directories listed, that when searching through the **/dev/slan** directory, if a file is encountered whose major/minor devices and file system identifier match those of the file descriptor argument to **ttyname**, this device name should be considered a match, and that neither **/dev/dsk** nor **/dev/rdsk** need be searched.

FILES

/etc/ttysrch

SEE ALSO

clone(7), **ttymap**(1M), **ttyname**(3C)

NAME

`unistd` – header file for symbolic constants

SYNOPSIS

`#include <unistd.h>`

DESCRIPTION

The **`unistd.h`** header file defines the symbolic constants and structures not already defined or declared in some other header file. The contents of this file are shown below.

The following symbolic constants are defined for the **`access`** function [see **`access`**(2)]:

`R_OK`	Test for read permission.
`W_OK`	Test for write permission.
`X_OK`	Test for execute (search) permission.
`F_OK`	Test for existence of file.

The constants `F_OK`, `R_OK`, `W_OK` and `X_OK` and the expressions `R_OK | W_OK`, `R_OK | X_OK` and `R_OK | W_OK | X_OK` all have distinct values.

Declares the constant

`NULL`	null pointer

The following symbolic constants are defined for the **`lockf`** function [see **`lockf`**(3C)]:

`F_ULOCK`	Unlock a previously locked region.
`F_LOCK`	Lock a region for exclusive use.
`F_TLOCK`	Test and lock a region for exclusive use.
`F_TEST`	Test a region for other processes locks.

The following symbolic constants are defined for the **`lseek`** [see **`lseek`**(2)] and **`fcntl`** [see **`fcntl`**(2)] functions (they have distinct values):

`SEEK_SET`	Set file offset to *offset*.
`SEEK_CUR`	Set file offset to current plus *offset*.
`SEEK_END`	Set file offset to `EOF` plus *offset*.

The following symbolic constants are defined (with fixed values):

`_POSIX_VERSION`	Integer value indicating version of the POSIX standard.
`_XOPEN_VERSION`	Integer value indicating version of the XPG to which system is compliant.

The following symbolic constants are defined to indicate that the option is present:

`_POSIX_JOB_CONTROL`	Implementation supports job control.
`_POSIX_SAVED_IDS`	The **`exec`** functions [see **`exec`**(2)] save the effective user and group.
`_POSIX_VDISABLE`	Terminal special characters defined in **`termios.h`** [see **`termio`**(7)] can be disabled using this character.

The following symbolic constants are defined for **sysconf** [see **sysconf**(3C)]:

```
_SC_ARG_MAX
_SC_CHILD_MAX
_SC_CLK_TCK
_SC_JOB_CONTROL
_SC_LOGNAME_MAX
_SC_NGROUPS_MAX
_SC_OPEN_MAX
_SC_PAGESIZE
_SC_PASS_MAX
_SC_SAVED_IDS
_SC_VERSION
_SC_XOPEN_VERSION
```

The following symbolic constants are defined for **pathconf** [see **fpathconf**(2)]:

```
_PC_CHOWN_RESTRICTED
_PC_LINK_MAX
_PC_MAX_CANON
_PC_MAX_INPUT
_PC_NAME_MAX
_PC_NO_TRUNC
_PC_PATH_MAX
_PC_PIPE_BUF
_PC_VDISABLE
```

The following symbolic constants are defined for file streams:

STDIN_FILENO	File number of **stdin**. It is 0.
STDOUT_FILENO	File number of **stout**. It is 1.
STDERR_FILENO	File number of **stderr**. It is 2.

The following pathnames are defined:

GF_PATH	Pathname of the group file.
PF_PATH	Pathname of the **passwd** file.

SEE ALSO

access(2), **exec**(2), **fcntl**(2), **fpathconf**(2), **group**(4), **lseek**(2), **passwd**(4), **sysconf**(3C), **termio**(7), **termios**(2)

NOTES

The following values for constants are defined for this release of System V:

_POSIX_VERSION	198808L
_XOPEN_VERSION	3

NAME

updaters – configuration file for Network Information Service (NIS) updating

SYNOPSIS

/var/yp/updaters

DESCRIPTION

The file **/var/yp/updaters** is a makefile [see **make**(1)] which is used for updating NIS databases. Databases can only be updated in a secure network, that is, one that has a **publickey**(4) database. Each entry in the file is a make target for a particular NIS database. For example, if there is a NIS database named **publickey.byname** that can be updated, there should be a **make** target named **publickey.byname** in the **updaters** file with the command to update the file.

The information necessary to make the update is passed to the update command through standard input. The information passed is described below (all items are followed by a NEWLINE, except for the actual bytes of key and actual bytes of date).

network name of client wishing to make the update (a string)

kind of update (an integer)

number of bytes in key (an integer)

actual bytes of key

number of bytes in data (an integer)

actual bytes of data

After getting this information through standard input, the command to update the particular database should decide whether the user is allowed to make the change. If not, it should exit with the status **YPERR_ACCESS**. If the user is allowed to make the change, the command should make the change and exit with a status of zero. If there are any errors that may prevent the updater from making the change, it should exit with the status that matches a valid NIS error code described in **<rpcsvc/ypclnt.h>**.

FILES

/var/yp/updaters

SEE ALSO

make(1), **publickey**(4), **ypupdate**(3N), **ypupdated**(1M)

NAME

utmp, wtmp – utmp and wtmp entry formats

SYNOPSIS

```
#include <utmp.h>
```

DESCRIPTION

These files, which hold user and accounting information for such commands as **who**, **write**, and **login**, have the following structure, defined in **utmp.h**:

```
#define UTMP_FILE   "/var/adm/utmp"
#define WTMP_FILE   "/var/adm/wtmp"
#define ut_name     ut_user
```

The **utmp** structure includes the following members:

```
   char     ut_user[8];         /* user login name */
   char     ut_id[4];           /* /etc/inittab id (created by */
                                /* process that puts entry in utmp) */
   char     ut_line[12];        /* device name (console, lnxx) */
   short    ut_pid;             /* process id */
   short    ut_type;            /* type of entry */
   struct   exit_status {
      short  e_termination;     /* process termination status */
      short  e_exit;            /* process exit status */
   } ut_exit;                   /* exit status of a process
                                * marked as DEAD_PROCESS */
   time_t  ut_time;             /* time entry was made */

/*  Definitions for ut_type  */

#define EMPTY          0
#define RUN_LVL        1
#define BOOT_TIME      2
#define OLD_TIME       3
#define NEW_TIME       4
#define INIT_PROCESS   5        /* process spawned by "init" */
#define LOGIN_PROCESS  6        /* a "getty" process waiting for login */
#define USER_PROCESS   7        /* a user process */
#define DEAD_PROCESS   8
#define ACCOUNTING     9
#define UTMAXTYPE      ACCOUNTING  /* max legal value of ut_type */

/*  Below are special strings or formats used in the "ut_line" */
/*  field when  accounting for something other than a process.  */
/*  No string for the ut_line field should be no more than 11 chars +  */
/*  a null character in length.  */

#define RUNLVL_MSG      "run-level %c"
#define BOOT_MSG        "system boot"
#define OTIME_MSG       "old time"
#define NTIME_MSG       "new time"
```

FILES

`/var/adm/utmp`
`/var/adm/wtmp`

SEE ALSO

`getut`(3C), `login`(1), `utmpx`(4), `who`(1), `write`(1)

NAME

utmpx, **wtmpx** – utmpx and wtmpx entry formats

SYNOPSIS

```
#include <utmpx.h>
```

DESCRIPTION

utmpx(4) is an extended version of **utmp**(4).

These files, which hold user and accounting information for such commands as **who**, **write**, and **login**, have the following structure as defined by **utmpx.h**:

```
#define   UTMPX_FILE   "/var/adm/utmpx"
#define   WTMPX_FILE   "/var/adm/wtmpx"
#define   ut_name     ut_user
#define   ut_xtime    ut_tv.tv_sec
```

The **utmpx** structure includes the following members:

```
   char   ut_user[32];          /* user login name */
   char   ut_id[4];             /* inittab id */
   char   ut_line[32];          /* device name (console, lnxx) */
   pid_t  ut_pid;               /* process id */
   short  ut_type;              /* type of entry */
   struct exit_status {
   short  e_termination;     /* termination status */
   short  e_exit;             /* exit status */
   } ut_exit;                /* process termination/exit status */
   struct timeval {
   long  tv_sec;             /* seconds */
   long  tv_usec;            /* and microseconds */
   ut_tv;                       /* time entry was made */
   long   ut_session;           /* session ID, used for windowing */
   long   pad[5];               /* reserved for future use */
   short  ut_syslen;            /* significant length of ut_host */
                                /* including terminating null */
   char   ut_host[257];         /* remote host name */

/* Definitions for ut_type */

#define   EMPTY           0
#define   RUN_LVL         1
#define   BOOT_TIME       2
#define   OLD_TIME        3
#define   NEW_TIME        4
#define   INIT_PROCESS    5  /* Process spawned by "init" */
#define   LOGIN_PROCESS   6  /* A "getty" process waiting for login */
#define   USER_PROCESS    7  /* A user process */
#define   DEAD_PROCESS    8
#define   ACCOUNTING      9

#define   UTMAXTYPE  ACCOUNTING  /* Largest legal value of ut_type */
```

```
/* Below are special strings or formats used in the "ut_line" */
/* field when accounting for something other than a process. */
/* No string for the ut_line field should be more than 31 chars + */
/* a null character in length. */

#define   RUNLVL_MSG     "run-level %c"
#define   BOOT_MSG       "system boot"
#define   OTIME_MSG      "old time"
#define   NTIME_MSG      "new time"
#define   MOD_WIN        10
```

FILES

/var/adm/utmpx
/var/adm/wtmpx

SEE ALSO

getutx(3C), login(1), utmp(4), who(1), write(1)

NAME

uuencode – format of an encoded uuencode file

DESCRIPTION

Files output by **uuencode** consist of a header line, followed by a number of body lines, and a trailer line. **uudecode** ignores 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** followed by a space). **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 characters 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)

NAME

vfstab – table of file system defaults

SYNOPSIS

```
#include <sys/fstyp.h>
#include <sys/param.h>
#include <sys/vfstab.h>
```

DESCRIPTION

The file **/etc/vfstab** describes defaults for each file system. The information is in the following structure, defined in **sys/vfstab.h**:

```
struct vfstab {
        char    *vfs_special;
        char    *vfs_fsckdev;
        char    *vfs_mountp;
        char    *vfs_fstype;
        char    *vfs_fsckpass;
        char    *vfs_automnt;
        char    *vfs_mntopts;
        char    *vfs_macceiling;
};
```

The fields in the table are space-separated and show the block special or resource name, the raw device to **fsck**, the default mount directory, the name of the file system type, the number used by **fsck** to decide whether to check the file system automatically, whether the file system should be mounted automatically by **mountall**, the mount options, and the default file system level ceiling. If Enhanced Security is not installed, the field that displays the default file system level ceiling is not used. A '-' is used to indicate no entry in a field.

The **getvfsent**(3C) family of routines are used to read and write to **/etc/vfstab**.

NOTES

Do not store information in the **vfstab** file other than the fields described above; fields may be added to this file in future releases and are reserved for future use.

SEE ALSO

fsck(1M), **getvfsent**(3C), **mount**(1M), **setmnt**(1M)

NAME

Xwincmaps – XWIN color map file

DESCRIPTION

The server reads the **/usr/X/defaults/Xwincmaps** file to fill up the static color map. Each line has 'R', 'G', and 'B' values. There are several colormaps in the default **Xwincmaps** file, but custom colormaps can be created and added to this file. The server takes the first uncommented colormap (without '#' in the first column) as the valid colormap data.

The format of the colormap data is:

```
colormap type screen_num num_colors
RED_VAL, BLUE_VAL, GREEN_VAL,
RED_VAL, BLUE_VAL, GREEN_VAL,
 .
 .
 .
```

Where

colormap	key word. This is the same for all entries.
type	type of colormap. This can be **PseudoColor**, **StaticColor**, or **GrayScale**.
screen_num	screen number
num_colors	number of colors
RED_VAL	red value
GREEN_VAL	green value
BLUE_VAL	blue value

The number of *RED_VAL, BLUE_VAL, GREEN_VAL,* lines should be the same as *num_colors*.

USAGE

To use the colormap of your choice, add a **#** to the current definition and remove the **#** sign in front of the new definition line, for example,

```
colormap   StaticColor      0.0   16
```

To use a private copy of your color map, either of the following commands can be used.

```
olinit -- -cmap $HOME/mycmap

X -cmap $HOME/mycmap &
```

EXAMPLES

The following is equivalent to the default colormap on the XWIN server.

```
colormap      StaticColor    0.0     16
    0x0000,     0x0000,     0x0000,
    0xFFFF,     0xFFFF,     0xFFFF,
    0xAAAA,     0xAAAA,     0xAAAA,
    0x0000,     0x0000,     0xAAAA,
    0x0000,     0x0000,     0xFFFF,
    0x0000,     0xAAAA,     0xFFFF,
    0x0000,     0xFFFF,     0xFFFF,
    0x0000,     0xAAAA,     0x0000,
    0x0000,     0xFFFF,     0x0000,
    0xAAAA,     0xFFFF,     0x5555,
    0xAAAA,     0x5555,     0x0000,
    0xFFFF,     0xAAAA,     0x0000,
    0xFFFF,     0xFFFF,     0x0000,
    0xAAAA,     0x0000,     0xAAAA,
    0xFFFF,     0x0000,     0xFFFF,
    0xFFFF,     0x0000,     0x0000,

#colormap     StaticColor    0.0     16
    0x0000,     0x0000,     0x0000,
    0xffff,     0xffff,     0xffff,
    0xa699,     0xa699,     0xa699, # a bit darker than openlook gray
    0x0000,     0x0000,     0xaaaa, # navy
    0x0000,     0x5144,     0xffff, # a bit less saturated than blue
    0x0000,     0xaaaa,     0xffff, # sky blue
    0x0000,     0xe79d,     0xe79d, # a bit darker than openlook cyan
    0x0000,     0xaaaa,     0x0000, # lime green (openlook forestgreen)
    0x0000,     0xdf7d,     0x0000, # a bit darker than green
    0x2081,     0x8e38,     0x69a6, # sea green
    0xaaaa,     0x5555,     0x0000, # brown
    0xf3ce,     0x9e79,     0x0000, # a bit darker than openlook orange
    0xffff,     0xffff,     0x0000, # yellow
    0xaaaa,     0x0000,     0xaaaa, # violet
    0x9248,     0xcb2b,     0x9248, # wheat-like color
    0xffff,     0x0000,     0x0000, # red
```

NOTES

`PseudoColor` is not recommended for 16 colors.

SEE ALSO

`Xwinconfig`(4)

Xwinconfig (4)

NAME

Xwinconfig – XWIN configuration file

DESCRIPTION

The server reads the first uncommented line in the **Xwinconfig** file for information about the display, colormap, and the information needed by the display driver. All the required data is on one line.

To change the current configuration, use the **setvgamode** command. To change modes manually, see the section "Editing the Xwinconfig File" below. In either case, the user must have appropriate privileges.

The format of the data of the first line in **Xwinconfig** file is:

```
display class cmap "INFO for SDD" scr_num tty display_lib
```

Where

display	device
class	display class, for example, **VGA16**, **VGA256**, **EGA**, **8514**, **XGA**, and so on.
cmap	type of colormap, for example, **StaticColor** or **PseudoColor**
INFO_for_SDD	information passed to SDD, for example, info passed to **libvga16.so**. See USAGE for a detailed description of this field.
scr_num	screen number
tty	tty, for example, **/dev/console**
display_lib	display library to link at run time, for example, **libvga16.so** or **libvga256.so**

USAGE

The following two sample entries in the **Xwinconfig** file illustrate the file format. Each line has been split in two (with a \ at the end of the first line) for display here, in the **Xwinconfig** file, each is one line.

```
display VGA16 StaticColor "VGA STDVGA 640x480 16 9.75x7.32" \
     0 /dev/console /usr/X/lib/libvga16.so
display VGA256 StaticColor "ET4000 MULTISYNC 1024x768 9.75x7.32" \
     0 /dev/console /usr/X/lib/libvga16.so
```

Description of the various fields:

1st field	type of device (currently, always **display**)
2nd field	video class (for example, **VGA16**, **VGA256**, **XGA**, **8514**, and so on)
3rd field	color class (always **StaticColor** for VGA16, **PseudoColor** for 256 and other high performance boards, and **GrayScale** for monochrome)

4th field	information passed to the "display driver" which is linked at run time. The format of this string is dependent on the display driver and varies for different display boards. Entries for both 16 and 256 colors can reside in the same configuration file. For example:

```
"ET4000 MULTISYNC 800x600 16 10.0x9.0"
```

`ET4000`	vendor or chipset identification.
`MULTISYNC`	type of monitor
`800x600`	resolution (width x height) in pixels
`16`	number of colors. For a 256 color driver, this field is omitted, because it supports only 256 colors. For a 16 color driver, this field may be `2`, `4`, or `16`. For monochrome, this field must be `2`.
`10.0x9.0`	monitor size (width x height) in inches

5th field	display number
6th field	the actual device (in most cases **/dev/console**)
7th field	the display driver

Editing the Xwinconfig file

For the advance user, editing the **Xwinconfig** file manually may be more flexible. The first uncommented line (without a **#** in the first column) is treated as a valid entry in the file.

To use a private copy of a configuration file, either of the following commands can be used:

```
olinit - - -config $HOME/mycfg
X -config $HOME/mycfg &
```

To return to the default mode (640x480, 16 colors), run the **setvgamode** command with the **-default** option.

Examples

This example illustrates how to manually set up files for a particular mode. The display is a Tseng Labs ET4000 based board and the mode is 1024x768 mode with 256 colors.

```
cp /usr/X/lib/vgainit/et4k_256i.so /usr/X/lib/libv256i.so.1
cp /usr/X/lib/vgainit/et4k.256cfg /usr/X/defaults/Xwinconfig
```

Edit the **/usr/X/defaults/Xwinconfig** file and remove the **#** sign in the first column of the line that has the 1024x768 entry. Make sure that all lines in the file before this entry are commented, as the first line without a **#** in the 1st column is treated as the active entry.

The procedure is the same for 16 modes. (that is, copy the **et4k_16i.so** and **et4k.16cfg** files).

The following example demonstrates creating a private copy of the config files, (avoids editing the system **Xwinconfig** file):

```
cp /usr/X/lib/vgainit/et4k.16cfg $HOME/cfg16
cp /usr/X/lib/vgainit/et4k.256cfg $HOME/cfg256
```

Now to run the server, either command can be used.

```
X -config $HOME/cfg16

X -config $HOME/cfg256
```

The standard VGA 640x480 mode is supported by all the 16 color SDDs. To find out the various entries provided by a particular SDD, follow the instructions above and give an invalid entry for the first field of the SDD information (for example, **foo** rather than **ET4000**). The following example prints out all the modes supported by that SDD.

```
display VGA16 StaticColor "foo STDVGA 640x480 16 10.0x9.0" \
     0 /dev/console /usr/X/lib/libvga16.so
```

Files

/usr/X/adm/setvgamode
/usr/X/defaults/Xwinconfig
/usr/X/defaults/Xwinconfig.ini

SEE ALSO

setvgamode(1M), **Xwincmaps**(4)

NAME

`Xwinfont` — XWIN font configuration and defaults file (scalable and bitmapped)

SYNOPSIS

`/usr/X/defaults/Xwinfont`

DESCRIPTION

The **`Xwinfont`** file specifies options concerning font rendering to the X server and allows several different font renders, both bitmapped and scalable, to be used. For example, it contains font configuration information showing the library to use for a renderer, the filename suffix for the renderer's font files, and the default pointsize to use for a scalable font if none is specified by an application.

The design of the font interface in XWIN provides the flexibility to add renderers to the X server without any code changes if the renderer uses the interface described in the document *Porting the XWIN Server*; this is done using dynamic shared libraries. This configuration file contains the information needed to use any such renderer—no changes to the server are required.

This file can also be used to tune font rendering in the server by allowing certain runtime combinations to be easily changed—for example, tuning for memory versus speed tradeoffs.

The format of the entries in the **`Xwinfont`** file is a simple keyword-value pair, with each element separated by an **`=`**. This is the same format as the X11R5 Font Server configuration file. The following types of values are supported:

cardinal	a non-negative number; generally used for specifying sizes
boolean	`1`, `y`, or `t` shows a feature present, `0`, `n`, or `f` shows a feature absent; used for yes/no or true/false possibilities.
string	an ASCII string
list	a list of *cardinal*s or *string*s separated by commas

Comments can be inserted in the file with lines that start with a **`#`**. Invalid options are logged to stderr (**`$HOME/.oliniterr`**) and are skipped. Some options are general options and affect all renderers; others are renderer-specific.

Renderer-specific options begin with a line containing **`startrenderer=`***suffix*, followed by a variable number of lines containing information specific to that renderer, and ended by either another line containing **`startrenderer=`***suffix* or the end of the file. *Suffix* is the font filename suffix the renderer uses to recognize files that it can render. Valid suffixes and their meanings are bitmap distribution format (**`bdf`**), server natural format (**`snf`**), portable compiled format (**`pcf`**), PostScript™ Type 1 format (**`pfa`** and **`pfb`**), Bitstream Speedo format (**`spd`**), and Folio (TypeScaler™) F3 format (**`f3b`**).

If several suffixes are valid for a renderer, as when the font files might have several valid styles of names, multiple **`startrenderer`** lines should be supplied, each with a valid suffix; the same options can be defined for each suffix. For example, the Type 1 outline font files for use with the Adobe Type Manager™ renderer can have, among others, suffixes of **`pfa`** or **`pfb`**.

The **snf** format is the default supported bitmap renderer and need not be explicitly defined. Its shared library is **libbitmap.so** in **/usr/X/lib**.

General Font Configuration Options

The options that apply to all renderers or to the X server appear first in the file, before the renderer-specific options.

fontpath=*listofstring*

The X server's default font path. This is a list of font path elements that the server will use in resolving **OpenFont** and **ListFonts** requests. It contains a comma separated list of string entries. If a full pathname is not given, the server will check the **XWINHOME** environment variable for the location of the default installed tree, and will prepend that path to each partial path. Font server names for fontpaths contain a colon separating the font server name from the path within the font server. The environment variable **XWINFONTPATH** or a server command line option (**-fp**) will override this option.

cachesize=*cardinal*

The **cachesize** option is the maximum value that the font cache can grow to without causing a font to be freed from the cache before inserting a new font. This value is expressed in K bytes and defaults to 800; it cannot be set lower than 256. The space is not preallocated for the cache. The cache will be allowed to grow to this value; if the maximum value is never reached, the freeing of fonts from the cache will not take place. Fonts are freed from the cache when they have been closed by all processes and users that had them open, or when a font is selected by the cache free routines to keep the maximum cache allocated for fonts within a maximum size. This maximum size can temporarily be exceeded if a font is open and more space is needed for the font glyphs; however, the space will be reduced to below the maximum cachesize at the first available opportunity. The font that caused the cache to overflow its maximum value will be chosen to free first, provided it is not the current font in use.

mincachesize=*cardinal*

This value is the low water mark for the cache. The value is expressed in units of K bytes. The cache starts allocating in non-pagesize chunks when the low water mark is reached. The default value is 90% of **cachesize**. The options

```
cachesize=500
mincachesize=450
```

would set a maximum font cache size of 500K bytes of memory and a low-water mark of 450K bytes of memory.

derived-instance-pointsizes=*list*

Applications may already exist that do not use scalable names, yet use the **ListFonts** protocol request with some pattern and expect to receive a list of fonts that specify a collection of point and pixel sizes. It was strongly encouraged by the X Logical Font Description Conventions that server vendors provide a mechanism for including in each scalable name a list of specific derived instance names for use by these applications. (A derived instance is the result of replacing scalable fields with values to yield a font name that could actually be produced from the font source.)

This option contains a list of cardinal values for point sizes that should be generated for scalable fonts to satisfy such requests. There is no default value if this is not specified in the file; the value placed in the file when outline fonts are installed is **`10,12,14`**. This option is not used directly by the X server; its value can be specified as the value of the **`DERIVED_INSTANCE_PS`** environment variable of the **`mkfontscale`** program, which creates the XLFD names in the **`fonts.scale`** in directories containing outline fonts (one XLFD name is created for each specified size, as well as for size 0, which indicates a scalable name). The Font Setup application of the UNIX Desktop runs the **`mkfontscale`** program, with the environment variable thus set, when the Actions / Integrity Check button menu item is chosen. The **`mkfontdir`** program creates the **`fonts.dir`** file from the **`fonts.scale`** file (this is also run by the Integrity Check menu button item); the X server reads the **`fonts.dir`** file to determine what fonts and derived instances are available.

Renderer-Specific Configuration Options

A renderer-specific configuration option is an option that only applies to the renderer used for font files who suffix was named in the most recent **`star-trenderer`** option. All renderer-specific options must be specified in the entries that follow a **`startrenderer=`***suffix*. Duplicate entries, for the same suffix, later in the file will override any previous entries; only the last set of renderer options in a file for any given named renderer are set. All options for a given renderer are grouped together between the **`startrenderer`** and the next **`startrenderer`** in the file (or the end of the file).

There are both public and private renderer options allowed. Renderer public options are defined below. Renderer private options are options that are unique to a specific renderer that might be added to the X server by OEMs and are defined and checked by the renderering library. Any unrecognized renderer options are assumed to be private options and are not parsed directly by the configuration file initialization routines (other than to set a flag to the renderer that private options were encountered). The renderer must check this flag for possible private options and for parsing these options. An example of an option that might be useful and private to a given renderer would be a list of glyphs to preallocate or prerender for a given font or for all fonts. How to specify the list would be renderer-specific.

`startrenderer=`*suffix*
: Start defining renderer-specific options for a renderer whose font files have the file name suffix *suffix*.

`use-renderer=`*boolean*
: The **`use-renderer`** option shows if this particular renderer should be used by the X server. The default, if the renderer is specified, and the **`use-renderer`** field is missing, is to include the rendering library. The server command line option **`-renderer`** *+suffix* or **`-renderer`** *–suffix* will override this option.

font-type=*string*
The **font-type** option denotes the type of rendering supported by this renderer. Valid values are **bitmap**, **scalable** and **both**. If both **bitmap** and **scalable** are valid, then the value **both** should be used. The default type is **scalable**.

sharedlib-filename=*string*
Contains the name and location of the font rendering library to load for this font format type. If the name is not a complete pathname, then the library will be searched for using the **lib** directory of the **XWINHOME** environment variable. This field will be used when the rendering library needs to be loaded. If the library is not found, an error will be returned when a font from that library is needed. This is a **mandatory** field. It is required since a renderer could be added after the Desktop product is released, and a full library specification in this field will enable the server to handle new font rendering libraries that conform to the specification outlined in *Porting the XWIN Server*. If more than one suffix applies to a single library, then multiple **startrenderer** entries and **sharedlib-filename** entries should be used.

defaultpoint=*cardinal*
Contains the default point size to be used in resolving scalable names that do not specify a point size or a pixel size in the request. This will be used to generate a derived instance of the font, if no other derived instance point sizes are given. If not specified, a compiled in default point size will be used (12).

preallocate-glyphs=*cardinal*
Rendering all the glyphs at open time requires that space be allocated at open as well. This will have a longer initialization time for the font as the glyphs are scaled. The other approach scales the character the first time it is accessed for display.

If glyphs are preallocated but not prerendered, this ensures enough space exists to render the entire font, and reserves it even though some glyphs may never get rendered.

A renderer-specific configuration option (**preallocate-glyphs**=*string*), determines how much space for the glyphs in a font should be preallocated. Value should be a *cardinal* value that specifies a number reflecting a percentage of the entire font to preallocate (that is, `10` for 10%). If the value is `0`, no glyphs should be preallocated. If the value is `100`, all the glyphs in each font should be preallocated; otherwise a value can be given that will represent the percentage of the total font glyph size that should be preallocated. The default for bitmap fonts, if not specified, is to preallocate the space for all the glyphs when the font is read.

prerender-glyphs=*boolean*
Another option (**prerender-glyphs**=*boolean*) specifies whether all the glyphs in the font should be rendered at open time (**y**), or as each glyph is used in a blitting routine (**n**). The default for bitmap fonts is they are already rendered. The default for scalable fonts is not to render the

glyph until it is needed, unless some renderer-specific private option overrides this. (For Adobe Type Manager, when it is installed, this option is set in the file to **y**, as this has proven to more efficient, given the information that the X server must get from the font for each character when just a single character is rendered.)

Other Options

Additional renderer-specific options are available. However, they should not be changed by a user or the owner of the machine, just an expert administrator or OEM developer who is monitoring font-related server performance.

preload-renderer=*boolean*
: This option will determine when the rendering library should be opened and the symbols resolved. If **preload-renderer** is set to false, the loading of the font renderer's library will be deferred until a font is needed. This will conserve the memory required by the library until it is needed, and will be the default for scalable renderers. If **preload-renderer** is true, the library will be loaded at server start up. This is the default for the bitmap based library containing the default cursor and text fonts.

free-renderer=*boolean*
: This option will determine when the rendering library will be closed and consequently when the memory used by the library is released. If set to true, the library will be closed when the last font is closed. This will free the space, but if another font from the library is opened later, it might cause a delay to reload the library. This is a performance/memory tradeoff option and will be helpful in doing internal performance evaluations. The default is set to false.

alloc-units=*cardinal*
: Indicates the minimum pagesize chunks that should be allocated for this renderer when allocating space for font glyphs. The default value is one pagesize chunk.

The **alloc-units** are saved in the font structure. This will allow renderers who wish to supply **alloc-units** by using renderer private options to override any general value.

download-glyphs=*string*
: Indicates if glyphs of this renderer should be downloaded to device dependent code. Possible values are **none**, **fixed** or **all**. The value **fixed** indicates only fixed width fonts should be downloaded. If **download-glyphs** is **none**, fonts for this renderer will not be downloaded. If set to **all**, then all fonts for this renderer will be downloaded.

download-height=*cardinal*
: This is the maximum height for glyphs in fonts that can be downloaded. Fonts containing any glyphs larger than height will not be downloaded.

download-width=*cardinal*
: This is the maximum width for glyphs in fonts that can be downloaded. Fonts containing any glyphs wider than width will not be downloaded.

download-maxchars=*cardinal*
If fonts are downloadable, then this is the maximum number of glyphs in fonts that can be downloaded. Fonts containing more characters than **download-maxchars** will not be downloaded.

EXAMPLES

The following options allow the Adobe Type Manager renderer to be used:

```
derived-instance-pointsizes=10,12,14
#
fontpath= lib/fonts/misc/,lib/fonts/Xol/,lib/fonts/75dpi/, \
lib/fonts/100dpi,lib/fonts/type1,lib/fonts/mitType1
#
startrenderer=pfa
prerender-glyphs=t
sharedlib-filename=libatm.so
#
startrenderer=pfb
use-renderer=t
prerender-glyphs=t
sharedlib-filename=libatm.so
```

The \ at the end of the font path line is just notation showing this line is longer than can be printed here; the \ is not parsed.

NOTES

Partial names that contain a colon are reserved for future font server names.

SEE ALSO

mkfontscale(1), **mkfontdir**(1)

NAME

ypfiles – the Network Information Service (NIS) database and directory structure

DESCRIPTION

The NIS network lookup service 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, one has the filename extension **.pag** and the other has the filename extension **.dir**. For instance, the database named **publickey**, is implemented by the pair of files **publickey.pag** and **publickey.dir**.

A **dbm** database served by the NIS is called a NIS *map*. A NIS *ypdomain* 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 NIS serves — if the map exists in a given domain, and a client asks about it, the NIS 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 privileged user's **crontab** file on each server. More information on this topic is in **ypxfr**(1M).

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**(1M) generates both key-value pairs automatically. A map that does not contain both key-value pairs can be served by the NIS, but the **ypserv** process will not be able to return values for "Get order number" or "Get master name" requests. See **ypserv**(1M). 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, extra command line switches must be set when it is run.

NIS maps must be generated and modified only at the master server. They are copied to the slaves using **ypxfr**(1M) to avoid potential byte-ordering problems among 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**(1M).

All NIS maps have entries in **/var/yp/aliases**. Each entry includes the map name and map nickname. The map name and nickname may be the same depending on the filesystem limitation of the length of filenames.

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**, see **ypmake**(1M). All Sun-supplied maps have entries in **/var/yp/Makefile**; if a NIS map is added, edit this file to support the new map. The makefile uses **makedbm**(1M) to generate the NIS map on the master, and

yppush(1M) 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**(1M).

FILES

```
/var/yp
/var/yp/aliases
/var/yp/Makefile
```

SEE ALSO

dbm(3), **makedbm**(1M), **publickey**(4), **ypinit**(1M), **ypmake**(1M), **yppoll**(1M), **yppush**(1M), **ypserv**(1M), **ypxfr**(1M)

NAME

`intro` – introduction to miscellany

DESCRIPTION

This section describes miscellaneous facilities such as macro packages, character set tables, and so forth.

ascii (5)

NAME

`ascii` – map of ASCII character set

DESCRIPTION

This is a map of the ASCII character set, giving both octal and hexadecimal equivalents of each character.

```
|000 nul|001 soh|002 stx|003 etx|004 eot|005 enq|006 ack|007 bel|
|010 bs |011 ht |012 nl |013 vt |014 np |015 cr |016 so |017 si |
|020 dle|021 dc1|022 dc2|023 dc3|024 dc4|025 nak|026 syn|027 etb|
|030 can|031 em |032 sub|033 esc|034 fs |035 gs |036 rs |037 us |
|040 sp |041  ! |042  " |043  # |044  $ |045  % |046  & |047  ' |
|050  ( |051  ) |052  * |053  + |054  , |055  - |056  . |057  / |
|060  0 |061  1 |062  2 |063  3 |064  4 |065  5 |066  6 |067  7 |
|070  8 |071  9 |072  : |073  ; |074  < |075  = |076  > |077  ? |
|100  @ |101  A |102  B |103  C |104  D |105  E |106  F |107  G |
|110  H |111  I |112  J |113  K |114  L |115  M |116  N |117  O |
|120  P |121  Q |122  R |123  S |124  T |125  U |126  V |127  W |
|130  X |131  Y |132  Z |133  [ |134  \ |135  ] |136  ^ |137  _ |
|140  ` |141  a |142  b |143  c |144  d |145  e |146  f |147  g |
|150  h |151  i |152  j |153  k |154  l |155  m |156  n |157  o |
|160  p |161  q |162  r |163  s |164  t |165  u |166  v |167  w |
|170  x |171  y |172  z |173  { |174  | |175  } |176  ~ |177 del|
```

```
| 00 nul| 01 soh| 02 stx| 03 etx| 04 eot| 05 enq| 06 ack| 07 bel|
| 08 bs | 09 ht | 0a nl | 0b vt | 0c np | 0d cr | 0e so | 0f si |
| 10 dle| 11 dc1| 12 dc2| 13 dc3| 14 dc4| 15 nak| 16 syn| 17 etb|
| 18 can| 19 em | 1a sub| 1b esc| 1c fs | 1d gs | 1e rs | 1f us |
| 20 sp | 21  ! | 22  " | 23  # | 24  $ | 25  % | 26  & | 27  ' |
| 28  ( | 29  ) | 2a  * | 2b  + | 2c  , | 2d  - | 2e  . | 2f  / |
| 30  0 | 31  1 | 32  2 | 33  3 | 34  4 | 35  5 | 36  6 | 37  7 |
| 38  8 | 39  9 | 3a  : | 3b  ; | 3c  < | 3d  = | 3e  > | 3f  ? |
| 40  @ | 41  A | 42  B | 43  C | 44  D | 45  E | 46  F | 47  G |
| 48  H | 49  I | 4a  J | 4b  K | 4c  L | 4d  M | 4e  N | 4f  O |
| 50  P | 51  Q | 52  R | 53  S | 54  T | 55  U | 56  V | 57  W |
| 58  X | 59  Y | 5a  Z | 5b  [ | 5c  \ | 5d  ] | 5e  ^ | 5f  _ |
| 60  ` | 61  a | 62  b | 63  c | 64  d | 65  e | 66  f | 67  g |
| 68  h | 69  i | 6a  j | 6b  k | 6c  l | 6d  m | 6e  n | 6f  o |
| 70  p | 71  q | 72  r | 73  s | 74  t | 75  u | 76  v | 77  w |
| 78  x | 79  y | 7a  z | 7b  { | 7c  | | 7d  } | 7e  ~ | 7f del|
```

NAME

environ – user environment

DESCRIPTION

When a process begins execution, **exec** routines make available an array of strings called the environment [see **exec**(2)]. By convention, these strings have the form *variable=value*, for example, **PATH=/sbin:/usr/sbin**. These environmental variables provide a way to make information about a program's environment available to programs. The following environmental variables can be used by applications and are expected to be set in the target runtime environment.

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

LANG The program's locale. Locales consist of files that describe the conventions appropriate to some nationality, culture, and language. Generally, users determine which files are selected by manipulating the environment variables described below. For background, see **setlocale**(3C).

Locales are partitioned into categories **LC_COLLATE**, **LC_CTYPE**, **LC_MESSAGES**, **LC_MONETARY**, **LC_NUMERIC**, and **LC_TIME** (see below for what the categories control). Each category has a corresponding environment variable that the user can set to specify that category's locale:

LC_CTYPE=fr[ancais]

The **LANG** environment variable is searched if the environment variable for a category is unset or empty:

LANG=fr
LC_COLLATE=de[utsche]

sets all the categories but **LC_COLLATE** to French. If **LANG** is unset or empty, the default **C** locale is used.

LC_COLLATE specifies the collation order used. The information for this category is stored in a database created by the **colltbl**(1M) command. This environment variable affects **sort**(1), **strcoll**(3C), and **strxfrm**(3C).

LC_CTYPE specifies character classification, character conversion, and widths of multibyte characters. The information for this category is stored in a database created by the **chrtbl**(1M) or **wchrtbl**(1M) commands. The default **C** locale uses the 7-bit US ASCII character set. This environment variable affects many commands and functions, among them, **cat**(1), **ed**(1), **ls**(1), **vi**(1), **ctype**(3C), and **mbchar**(3C),

LC_MESSAGES specifies the message database used. A command or application may have French and German message databases, for example. Message databases are created by the **mkmsgs**(1) or **gencat**(1) commands. This environment variable affects **gettxt**(1),

srchtxt(1), **catgets**(3C), and **gettxt**(3C), and every command that generates locale-specific output messages.

LC_MONETARY specifies the monetary symbols and delimiters used. The information for this category is stored in a database created by the **montbl**(1M) command. This environment variable affects **localeconv**(3C).

LC_NUMERIC specifies the decimal and thousands delimiters. The information for this category is stored in a database created by the **chrtbl**(1M) or **wchrtbl**(1M) commands. The default **C** locale uses a period (**.**) as the decimal delimiter and no thousands delimiter. This environment variable affects **localeconv**(3C), **printf**(3S), **scanf**(3S), and **strtod**(3C).

LC_TIME specifies date and time formats. The information for this category is stored in a database specified in **strftime**(4). The default **C** locale uses US date and time formats. This environment variable affects many commands and functions, among them, **at**(1), **calendar**(1), **date**(1), **getdate**(3C), and **strftime**(3C).

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

SEV_LEVEL Defines severity levels and associates and prints strings with them in standard format error messages [see **addseverity**(3C), **fmtmsg**(1), and **fmtmsg**(3C)].

NETPATH A colon-separated list of network identifiers. A network identifier is a character string used by the Network Selection component of the system to provide application-specific default network search paths. A network identifier must consist of non-NULL characters and must have a length of at least 1. No maximum length is specified. Network identifiers are normally chosen by the system administrator. A network identifier is also the first field in any **/etc/netconfig** file entry. **NETPATH** thus provides a link into the **/etc/netconfig** file and the information about a network contained in that network's entry. **/etc/netconfig** is maintained by the system administrator. The library routines described in **getnetpath**(3N) access the **NETPATH** environment variable.

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

For example:

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

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

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

%N	The value of the *name* parameter passed to **catopen**.
%L	The value of **LANG**.
%l	The language element from **LANG**.
%t	The territory element from **LANG**.
%c	The codeset element from **LANG**.
%%	A single **%** character.

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

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

For example:

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

indicates to **catopen** that it should look for the requested message catalog in *name*, *name*.**cat**, and **/nlslib/$LANG/***name*.**cat**.

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

SHELL When the shell is invoked, it scans the environment for this name. If it is found and **rsh** is the filename part of its value, the shell becomes a restricted shell. The value of this variable should be specified with an absolute pathname. The variable is used by **make**(1), **ksh**(1), **sh**(1), and **vi**(1), among other commands.

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

TZ Time zone information. The contents of the environment variable named **TZ** are used by the functions **ctime**(3C), **localtime** [see **ctime**(3C)], **strftime**(3C), and **mktime**(3C) to override the default time zone. If the first character of **TZ** is a colon (**:**), the behavior is implementation-defined, otherwise **TZ** has the form:

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

Three or more bytes that are the designation for the standard (*std*) and daylight savings time (*dst*) time zones. Only *std* is required, if *dst* is missing, then daylight savings time does not apply in this locale. Upper- and lowercase letters are allowed. Any characters except a leading colon (**:**), digits, a comma (**,**), a minus (**-**), or a plus (**+**) are allowed.

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

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

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

start / time, end / time

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

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

J*n*	The Julian day n ($1 \le n \le 365$). Leap days are not counted. That is, in all years, February 28 is day 59 and March 1 is day 60. It is impossible to refer to the occasional February 29.
n	The zero-based Julian day ($0 \le n \le 365$). Leap days are counted, and it is possible to refer to February 29.
M*m.n.d*	The d^{th} day, ($0 \le d \le 6$) of week n of month m of the year ($1 \le n \le 5$, $1 \le m \le 12$), where week 5 means "the last d-day in month m" which may occur in either the fourth or the fifth week). Week 1 is the first week in which the d^{th} day occurs. Day zero is Sunday.

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

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

Further names may be placed in the environment by the **export** command and *name=value* arguments in **sh**(1), or by **exec**(2). It is unwise to conflict with certain shell variables that are frequently exported by **.profile** files: **MAIL**, **PS1**, **PS2**, **IFS** [see **profile**(4)].

SEE ALSO

addseverity(3C), **cat**(1), **catgets**(3C), **catopen**(3C), **chrtbl**(1M), **colltbl**(1M), **ctime**(3C), **ctype**(3C), **date**(1), **ed**(1), **exec**(2), **fmtmsg**(1), **fmtmsg**(3C), **gencat**(1), **getdate**(3C), **getnetpath**(3N), **gettxt**(1), **gettxt**(3C), **localeconv**(3C), **login**(1), **ls**(1), **mbchar**(3C), **mkmsgs**(1), **mktime**(3C), **montbl**(1M), **netconfig**(4), **nice**(1), **nohup**(1), **passwd**(4), **printf**(3S), **profile**(4) **scanf**(3S), **setlocale**(3C), **sh**(1), **sort**(1), **srchtxt**(1), **strcoll**(3C), **strftime**(3C), **strftime**(4), **strtod**(3C), **strxfrm**(3C), **time**(1), **timezone**(4) **vi**(1), **wchrtbl**(1M)

NAME

eqnchar – (BSD) special character definitions for **eqn**

SYNOPSIS

eqn /usr/ucblib/pub/eqnchar [*file*] | **troff** [*options*]

neqn /usr/ucblib/pub/eqnchar [*file*] | **nroff** [*options*]

DESCRIPTION

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

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

FILES

/usr/ucblib/pub/eqnchar

SEE ALSO

eqn(1), **nroff**(1), **troff**(1)

NAME

eucioctl – generic interface to EUC handling tty drivers and modules

SYNOPSIS

```
#include <sys/eucioctl.h>

ioctl(int fd, I_STR, struct strioctl *sb);
```

DESCRIPTION

This interface is implemented in tty drivers and pushable STREAMS modules that handle EUC codes. It is intended as a generic interface for EUC handling, to eliminate an explosion of module-specific **ioctl** calls that would otherwise be necessary, and to provide uniformity in dealing with EUC code sets in the tty subsystem.

Several calls are defined. The first two calls take an argument, which is expected to be a pointer to an **eucioc** structure, defined in the header file **sys/eucioctl.h**:

```
struct eucioc {
      unsigned char eucw[4];
      unsigned char scrw[4];
};
typedef struct eucioc eucioc_t;
```

In all cases, these calls return non-zero on failure. Failure should be usually taken as an indication that the current driver, or line discipline module, does not support EUC, in which case **errno** will be set to **EINVAL**. For the **EUC_WSET** or **EUC_WGET** calls, **errno** will be set to **EPROTO** if **struct eucioc** argument is invalid.

EUC_WSET This call takes a pointer to an **eucioc** structure, and uses it to set the EUC line discipline's local definition for the code set widths to be used for subsequent operations. Within the stream, the line discipline may optionally notify other modules of this setting via **M_CTL** messages.

EUC_WGET This call takes a pointer to an **eucioc** structure, and returns in it the EUC code set widths currently in use by the EUC line discipline. It needs to be recognized only by line discipline modules.

The following calls take no arguments. They should only fail if the driver (at the bottom of the tty stream) does not recognize EUC codes. Drivers that support EUC, whether the stream contains modules that respond to the calls or not, will recognize the calls and acknowledge them. These calls are normally only interpreted by modules that have modes other than ASCII, and/or do some form of I/O conversion that normally prevents a program from receiving non-EUC characters in its byte stream. All of these calls, when received by modules, are passed down the tty stream, to be ultimately acknowledged by the tty driver.

EUC_MSAVE This call has no effect on modules that are currently in ASCII mode. Otherwise (i.e., for modules not in ASCII mode), the following actions are taken by all modules that recognize this call: (1) the current mode status is saved, (2) the mode is changed to ASCII mode immediately.

EUC_MREST	If a mode was saved via a previous **EUC_MSAVE** call, the saved mode is restored, and the saved state flag is cleared. If the mode was not previously saved, this call has no effect. (The exact semantics are somewhat dependent on the module, since some modules may respond to specific user requests to switch modes, even while a mode is being saved via **EUC_MSAVE**.)
EUC_IXLOFF	If a module is currently in a state where input conversion is being performed on the incoming byte stream, then input conversion is turned off, and the module's mode status is saved. If no input conversion is being performed, there is no effect on the module. The purpose of this call is to provide a way of insuring a pure byte stream to the program. The byte stream while input conversion is off is, of course, not guaranteed to be a stream of EUC characters. Turning off input conversion is roughly equivalent to the old concept of raw mode, if used in conjunction with **ICANON** off. It should normally not be used by applications.
EUC_IXLON	If a module previously saved its state and turned off input conversion, then input conversion is restored (i.e., turned back on); otherwise, there is no effect.
EUC_OXLOFF	In a manner similar to **EUC_IXLOFF**, any output conversion is turned off and the current mode status saved.
EUC_OXLON	In a manner similar to **EUC_IXLON**, any saved output conversion status is restored (i.e., output conversion is turned back on if previously turned off via **EUC_OXLOFF**).

FILES

/usr/include/sys/eucioctl.h

NOTES

Drivers and modules that support EUC should all respond appropriately to these calls, depending on their type. Line disciplines must respond to **EUC_WSET** and **EUC_WGET**, changing their current code set sizes to match **EUC_WSET** requests. All tty STREAMS modules that do any input or output conversion should recognize the other calls; modules that do no code set conversion are not required to recognize the calls, but must pass them through. Drivers that support EUC tty streams must all acknowledge the ON/OFF calls, whether the drivers themselves are affected or not, since these calls are purposely not acknowledged by modules which receive them; they are intended to be made available for affecting all modules in the whole stream.

Adherence to this protocol for all EUC handling modules is strongly encouraged in order to increase portability and language-independence of applications. These calls are intended as a small set of primitives to help reduce an anticipated plethora of module- and language-dependent operations.

NAME

`fcntl` – file control options

SYNOPSIS

`#include <sys/fcntl.h>`

DESCRIPTION

The **`fcntl.h`** header defines the following requests and arguments for use by the functions **`fcntl`** [see **`fcntl`**(2)] and **open** [see **open**(2)].

Values for *cmd* used by **`fcntl`** (the following values are unique):

`F_DUPFD`	Duplicate file descriptor
`F_GETFD`	Get file descriptor flags
`F_SETFD`	Set file descriptor flags
`F_GETFL`	Get file status flags
`F_SETFL`	Set file status flags
`F_GETLK`	Get record locking information
`F_SETLK`	Set record locking information
`F_SETLKW`	Set record locking information; wait if blocked

File descriptor flags used for **`fcntl`**:

`FD_CLOEXEC`	Close the file descriptor upon execution of an exec function [see **exec**(2)]

Values for **`l_type`** used for record locking with **`fcntl`** (the following values are unique):

`F_RDLCK`	Shared or read lock
`F_UNLCK`	Unlock
`F_WRLCK`	Exclusive or write lock

The following three sets of values are bitwise distinct:

Values for **`oflag`** used by **open**:

`O_CREAT`	Create file if it does not exist
`O_EXCL`	Exclusive use flag
`O_NOCTTY`	Do not assign controlling tty
`O_TRUNC`	Truncate flag

File status flags used for **open** and **`fcntl`**:

`O_APPEND`	Set append mode
`O_NDELAY`	Non-blocking mode
`O_NONBLOCK`	Non-blocking mode (POSIX)
`O_SYNC`	Synchronous writes

Mask for use with file access modes:

`O_ACCMODE`	Mask for file access modes

File access modes used for **open** and **`fcntl`**:

`O_RDONLY`	Open for reading only
`O_RDWR`	Open for reading and writing
`O_WRONLY`	Open for writing only

The structure `flock` describes a file lock. It includes the following members:

```
short    l_type;       /* Type of lock */
short    l_whence;     /* Flag for starting offset */
off_t    l_start;      /* Relative offset in bytes */
off_t    l_len;        /* Size; if 0 then until EOF */
long     l_sysid;      /* Returned with F_GETLK */
pid_t    l_pid;        /* Returned with F_GETLK */
long     l_pad         /* reserve area */
```

SEE ALSO

`creat`(2), `exec`(2), `fcntl`(2), `open`(2)

NAME

`font` – font description files for `troff` and `dpost`

SYNOPSIS

`troff -T` *ptty* . . .

DESCRIPTION

For each typesetter or printer type supported by `troff`(1) and available on this system, there is a directory containing files describing the device and its fonts. This directory is named `/usr/lib/font/dev`*ptty* where *ptty* is the abbreviated name of the typesetter or printer type. Currently supported devices are **`aps`** for the Autologic APS–5, **`post`** for PostScript printers, and **`i10`** for the Imagen Imprint 10 laser printer.

For a particular phototypesetter, *ptty*, the ASCII file *DESC* in the directory **`/usr/lib/font/dev`***ptty* describes its characteristics. Each line starts with a word identifying the characteristic which is followed by appropriate specifiers. Blank lines and lines beginning with a **#** are ignored.

The legal lines for *DESC* are:

`res` *num*	resolution of device in basic increments per inch
`hor` *num*	smallest unit of horizontal motion
`vert` *num*	smallest unit of vertical motion
`unitwidth` *num*	pointsize in which widths are specified
`sizescale` *num*	scaling for fractional pointsizes
`paperwidth` *num*	width of paper in basic increments
`paperlength` *num*	length of paper in basic increments
`biggestfont` *num*	maximum size of a font
`spare2` *num*	available for use
`sizes` *num num* . . .	list of pointsizes available on typesetter
`fonts` *num name* . . .	number of initial fonts followed by the names of the fonts. For example: fonts 4 R I B S
`charset`	this always comes last in the file and is on a line by itself. Following it is the list of special character names for this device. Names are separated by a space or a newline. The list can be as long as necessary. Names not in this list are not allowed in the font description files.

`Res` is the basic resolution of the device in increments per inch. **`Hor`** and **`vert`** describe the relationships between motions in the horizontal and vertical directions. If the device is capable of moving in single basic increments in both directions, both **`hor`** and **`vert`** would have values of 1. If the vertical motions only take place in multiples of two basic units while the horizontal motions take place in the basic increments, then **`hor`** would be 1, while **`vert`** would be 2. The **`unitwidth`** is the pointsize in which all width tables in the font description files are given. **`troff`**

automatically scales the widths from the **unitwidth** size to the pointsize it is working with. **Sizescale** is not currently used and is 1. **paperwidth** is the width of the paper in basic increments. The APS-5 is 6120 increments wide. **paperlength** is the length of a sheet of paper in the basic increments. **biggestfont** is the maximum number of characters on a font.

For each font supported by the phototypesetter, there is also an ASCII file with the same name as the font (e.g., **R**, **I 4**). The format for a font description file is:

name *name*	name of the font, such as **R** or **4**
internalname *name*	internal name of font
special	sets flag indicating that the font is special
ligatures *name* . . . **0**	Sets flag indicating font has ligatures. The list of ligatures follows and is terminated by a zero. Accepted ligatures are: **ff fi fl ffi ffl** .
spare1	available for use
spacewidth *num*	width of space if something other than 1/3 of — is desired as a space.
charset	The character set must come at the end. Each line following the word *charset* describes one character in the font. Each line has one of two formats: *name width kerning code* / *name* " — where *name* is either a single ASCII character or a special character name from the list found in *DESC*. The width is in basic increments. The kerning information is 1 if the character descends below the line, 2 if it rises above the letter 'a', and 3 if it both rises and descends. The kerning information for special characters is not used and so may be 0. The code is the number sent to the typesetter to produce the character. The second format is used to indicate that the character has more than one name. The double quote indicates that this name has the same values as the preceding line. The kerning and code fields are not used if the width field is a double quote character. The total number of different characters in this list should not be greater than the value of **biggestfont** in the DESC file (see above).

troff and its postprocessors like **dpost** read this information from binary files produced from the ASCII files by a program distributed with **troff** called **makedev**. For those with a need to know, a description of the format of these files follows:

The file *DESC.out* starts with the *dev* structure, defined by *dev.h*:

```
 /*
dev.h: characteristics of a typesetter
* /

struct dev {
unsigned short      filesize; /* number of bytes in file, */
                     /* excluding dev part */
short   res;         /* basic resolution in goobies/inch */
short   hor;         /* goobies horizontally */
short   vert;
short   unitwidth;   /* size at which widths are given*/
short   nfonts;      /* number fonts physically available */
short   nsizes;      /* number of pointsizes */
short   sizescale;   /* scaling for fractional pointsizes */
short   paperwidth;  /* max line length in units */
short   paperlength;  /* max paper length in units */
short   nchtab;      /* number of funny names in chtab */
short   lchname;     /* length of chname table */
short   biggestfont; /* max # of chars in a font */
short   spare2;
};
```

Filesize is just the size of everything in *DESC.out* excluding the *dev* structure. *Nfonts* is the number of different font positions available. *Nsizes* is the number of different pointsizes supported by this typesetter. *Nchtab* is the number of special character names. *Lchname* is the total number of characters, including nulls, needed to list all the special character names. At the end of the structure are two spares for later expansions.

Immediately following the *dev* structure are a number of tables. First is the *sizes* table, which contains *nsizes* + 1 shorts(a null at the end), describing the pointsizes of text available on this device. The second table is the *funny_char_index_table*. It contains indices into the table which follows it, the *funny_char_strings*. The indices point to the beginning of each special character name which is stored in the *funny_char_strings* table. The *funny_char_strings* table is *lchname* characters long, while the *funny_char_index_table* is *nchtab* shorts long.

Following the *dev* structure will occur *nfonts {font}.out* files, which are used to initialize the font positions. These *{font}.out* files, which also exist as separate files, begin with a *font* structure and then are followed by four character arrays:

```
struct  font {        /* characteristics of a font */
char    nwfont;       /* number of width entries */
char    specfont;     /* 1 == special font */
char    ligfont;      /* 1 == ligatures exist on this font */
char    spare1;       /* unused for now */
char    namefont[10]; /* name of this font, e.g., R */
char    intname[10];  /* internal name of font, in ASCII */
};
```

The *font* structure tells how many defined characters there are in the font, whether the font is a "special" font and if it contains ligatures. It also has the ASCII name of the font, which should match the name of the file it appears in, and the internal name of the font on the typesetting device (*intname*). The internal name is independent of the font position and name that **troff** knows about. For example, you might say mount **R** in position **4**, but when asking the typesetter to actually produce a character from the **R** font, the postprocessor which instructs the typesetter would use *intname*.

The first three character arrays are specific for the font and run in parallel. The first array, *widths*, contains the width of each character relative to *unitwidth*. *unitwidth* is defined in *DESC*. The second array, *kerning*, contains kerning information. If a character rises above the letter 'a', 02 is set. If it descends below the line, 01 is set. The third array, *codes*, contains the code that is sent to the typesetter to produce the character.

The fourth array is defined by the device description in *DESC*. It is the *font_index_table*. This table contains indices into the *width*, *kerning*, and *code* tables for each character. The order that characters appear in these three tables is arbitrary and changes from one font to the next. In order for **troff** to be able to translate from ASCII and the special character names to these arbitrary tables, the *font_index_table* is created with an order which is constant for each device. The number of entries in this table is 96 plus the number of special character names for this device. The value 96 is 128 - 32, the number of printable characters in the ASCII alphabet. To determine whether a normal ASCII character exists, **troff** takes the ASCII value of the character, subtracts 32, and looks in the *font_index_table*. If it finds a 0, the character is not defined in this font. If it finds anything else, that is the index into *widths*, *kerning*, and *codes* that describe that character.

To look up a special character name, for example **\(pl**, the mathematical plus sign, and determine whether it appears in a particular font or not, the following procedure is followed. A *counter* is set to 0 and an index to a special character name is picked out of the *counter'th* position in the *funny_char_index_table*. A string comparison is performed between *funny_char_strings [funny_char_index_table [counter]]* and the special character name, in our example **pl**, and if it matches, then **troff** refers to this character as (96 + *counter*). When it wants to determine whether a specific font supports this character, it looks in *font_index_table[(96+counter)]*, (see below), to see whether there is a 0, meaning the character does not appear in this font, or a number, which is the index into the *widths*, *kerning*, and *codes* tables.

Notice that since a value of 0 in the *font_index_table* indicates that a character does not exist, the 0th element of the *width*, *kerning*, and *codes* arrays are not used. For this reason the 0th element of the *width* array can be used for a special purpose, defining the width of a space for a font. Normally a space is defined by **troff** to be 1/3 of the width of the \(em character, but if the 0th element of the *width* array is non-zero, then that value is used for the width of a space.

SEE ALSO

dpost(1), **troff**(1), **troff**(5)

FILES

`/usr/lib/font/dev`*X*`/`**`DESC.out`**	description file for phototypesetter *X*
`/usr/lib/font/dev`*X*`/`*font*`.out`	*font* description files for phototypesetter *X*

iconv (5)

NAME

iconv – code set conversion tables

DESCRIPTION

The following code set conversions are supported:

Code Set Conversions Supported				
Code	Symbol	Target Code	Symbol	comment
ISO 646	646	ISO 8859-1	8859	US ASCII
ISO 646de	646de	ISO 8859-1	8859	German
ISO 646da	646da	ISO 8859-1	8859	Danish
ISO 646en	646en	ISO 8859-1	8859	English ASCII
ISO 646es	646es	ISO 8859-1	8859	Spanish
ISO 646fr	646fr	ISO 8859-1	8859	French
ISO 646it	646it	ISO 8859-1	8859	Italian
ISO 646sv	646sv	ISO 8859-1	8859	Swedish
ISO 8859-1	8859	ISO 646	646	7 bit ASCII
ISO 8859-1	8859	ISO 646de	646de	German
ISO 8859-1	8859	ISO 646da	646da	Danish
ISO 8859-1	8859	ISO 646en	646en	English ASCII
ISO 8859-1	8859	ISO 646es	646es	Spanish
ISO 8859-1	8859	ISO 646fr	646fr	French
ISO 8859-1	8859	ISO 646it	646it	Italian
ISO 8859-1	8859	ISO 646sv	646sv	Swedish

The conversions are performed according to the tables following. All values in the tables are given in octal.

ISO 646 (US ASCII) to ISO 8859-1

For the conversion of ISO 646 to ISO 8859-1 all characters in ISO 646 can be mapped unchanged to ISO 8859-1

ISO 646de (GERMAN) to ISO 8859-1

For the conversion of ISO 646de to ISO 8859-1 all characters not in the following table are mapped unchanged.

Conversions Performed	
ISO 646de	ISO 8859-1
100	247
133	304
134	326
135	334
173	344
174	366
175	374
176	337

ISO 646da (DANISH) to ISO 8859-1

For the conversion of ISO 646da to ISO 8859-1 all characters not in the following table are mapped unchanged.

Conversions Performed	
ISO 646da	ISO 8859-1
133	306
134	330
135	305
173	346
174	370
175	345

ISO 646en (ENGLISH ASCII) to ISO 8859-1

For the conversion of ISO 646en to ISO 8859-1 all characters not in the following table are mapped unchanged.

Conversions Performed	
ISO 646en	ISO 8859-1
043	243

ISO 646fr (FRENCH) to ISO 8859-1

For the conversion of ISO 646fr to ISO 8859-1 all characters not in the following table are mapped unchanged.

Conversions Performed	
ISO 646fr	ISO 8859-1
043	243
100	340
133	260
134	347
135	247
173	351
174	371
175	350
176	250

ISO 646it (ITALIAN) to ISO 8859-1

For the conversion of ISO 646it to ISO 8859-1 all characters not in the following table are mapped unchanged.

Conversions Performed	
ISO 646it	ISO 8859-1
043	243
100	247
133	260
134	347
135	351
140	371
173	340
174	362
175	350
176	354

ISO 646es (SPANISH) to ISO 8859-1

For the conversion of ISO 646es to ISO 8859-1 all characters not in the following table are mapped unchanged.

Conversions Performed	
ISO 646es	ISO 8859-1
100	247
133	241
134	321
135	277
173	260
174	361
175	347

ISO 646sv (SWEDISH) to ISO 8859-1

For the conversion of ISO 646sv to ISO 8859-1 all characters not in the following table are mapped unchanged.

Conversions Performed	
ISO 646sv	ISO 8859-1
100	311
133	304
134	326
135	305
136	334
140	351
173	344
174	366
175	345
176	374

ISO 8859-1 to ISO 646 (ASCII)

For the conversion of ISO 8859-1 to ISO 646 all characters not in the following table are mapped unchanged.

Converted to Underscore '_' (137)							
200	201	202	203	204	205	206	207
210	211	212	213	214	215	216	217
220	221	222	223	224	225	226	227
230	231	232	233	234	235	236	237
240	241	242	243	244	245	246	247
250	251	252	253	254	255	256	257
260	261	262	263	264	265	266	267
270	271	272	273	274	275	276	277
300	301	302	303	304	305	306	307
310	311	312	313	314	315	316	317
320	321	322	323	324	325	326	327
330	331	332	333	334	335	336	337
340	341	342	343	344	345	346	347
350	351	352	353	354	355	356	357
360	361	362	363	364	365	366	367
370	371	372	373	374	375	376	377

ISO 8859-1 to ISO 646de (GERMAN)

For the conversion of ISO 8859-1 to ISO 646de all characters not in the following tables are mapped unchanged.

Conversions Performed	
ISO 8859-1	ISO 646de
247	100
304	133
326	134
334	135
337	176
344	173
366	174
374	175

Converted to Underscore '_' (137)

100	133	134	135	173	174	175	176
200	201	202	203	204	205	206	207
210	211	212	213	214	215	216	217
220	221	222	223	224	225	226	227
230	231	232	233	234	235	236	237
240	241	242	243	244	245	246	
250	251	252	253	254	255	256	257
260	261	262	263	264	265	266	267
270	271	272	273	274	275	276	277
300	301	302	303		305	306	307
310	311	312	313	314	315	316	317
320	321	322	323	324	325		327
330	331	332	333		335	336	337
340	341	342	343		345	346	347
350	351	352	353	354	355	356	357
360	361	362	363	364	365		367
370	371	372	373		375	376	377

ISO 8859-1 to ISO 646da (DANISH)

For the conversion of ISO 8859-1 to ISO 646da all characters not in the following tables are mapped unchanged.

Conversions Performed	
ISO 8859-1	ISO 646da
305	135
306	133
330	134
345	175
346	173
370	174

Converted to Underscore '_' (137)

133	134	135	173	174	175	
200	201	202	203	204	205	206
210	211	212	213	214	215	216
220	221	222	223	224	225	226
230	231	232	233	234	235	236
240	241	242	243	244	245	246
250	251	252	253	254	255	256
260	261	262	263	264	265	266
270	271	272	273	274	275	276
300	301	302	303	304		
310	311	312	313	314	315	316
320	321	322	323	324	325	326
	331	332	333	334	335	336
340	341	342	343	344		
350	351	352	353	354	355	356
360	361	362	363	364	365	366
	371	372	373	374		376

ISO 8859-1 to ISO 646en (ENGLISH ASCII)

For the conversion of ISO 8859-1 to ISO 646en all characters not in the following tables are mapped unchanged.

Conversions Performed	
ISO 8859-1	ISO 646en
243	043

Converted to Underscore '_' (137)							
043							
200	201	202	203	204	205	206	207
210	211	212	213	214	215	216	217
220	221	222	223	224	225	226	227
230	231	232	233	234	235	236	237
240	241	242		244	245	246	247
250	251	252	253	254	255	256	257
260	261	262	263	264	265	266	267
270	271	272	273	274	275	276	277
300	301	302	303	304	305	306	307
310	311	312	313	314	315	316	317
320	321	322	323	324	325	326	327
330	331	332	333	334	335	336	337
340	341	342	343	344	345	346	347
350	351	352	353	354	355	356	357
360	361	362	363	364	365	366	367
370	371	372	373	374	375	376	377

ISO 8859-1 to ISO 646fr (FRENCH)

For the conversion of ISO 8859-1 to ISO 646fr all characters not in the following tables are mapped unchanged.

Conversions Performed	
ISO 8859-1	ISO 646fr
243	043
247	135
250	176
260	133
340	100
347	134
350	175
351	173
371	174

Converted to Underscore '_' (137)

043							
100	133	134	135	173	174	175	176
200	201	202	203	204	205	206	207
210	211	212	213	214	215	216	217
220	221	222	223	224	225	226	227
230	231	232	233	234	235	236	237
240	241	242		244	245	246	
	251	252	253	254	255	256	257
	261	262	263	264	265	266	267
270	271	272	273	274	275	276	277
300	301	302	303	304	305	306	307
310	311	312	313	314	315	316	317
320	321	322	323	324	325	326	327
330	331	332	333	334	335	336	337
	341	342	343	344	345	346	
		352	353	354	355	356	357
360	361	362	363	364	365	366	367
370		372	373	374	375	376	377

ISO 8859-1 to ISO 646it (ITALIAN)

For the conversion of ISO 8859-1 to ISO 646it all characters not in the following tables are mapped unchanged.

Conversions Performed	
ISO 8859-1	ISO 646it
243	043
247	100
260	133
340	173
347	134
350	175
351	135
354	176
362	174
371	140

Converted to Underscore '_' (137)

043							
100	133	134	135	173	174	175	176
200	201	202	203	204	205	206	207
210	211	212	213	214	215	216	217
220	221	222	223	224	225	226	227
230	231	232	233	234	235	236	237
240	241	242		244	245	246	
250	251	252	253	254	255	256	257
	261	262	263	264	265	266	267
270	271	272	273	274	275	276	277
300	301	302	303	304	305	306	307
310	311	312	313	314	315	316	317
320	321	322	323	324	325	326	327
330	331	332	333	334	335	336	337
	341	342	343	344	345	346	
		352	353	354	355	356	357
360	361		363	364	365	366	367
370		372	373	374	375	376	377

ISO 8859-1 to ISO 646es (SPANISH)

For the conversion of ISO 8859-1 to ISO 646es all characters not in the following tables are mapped unchanged.

Conversions Performed	
ISO 8859-1	ISO 646es
241	133
247	100
260	173
277	135
321	134
347	175
361	174

Converted to Underscore '_' (137)

100	133	134	135	173	174	175	
200	201	202	203	204	205	206	207
210	211	212	213	214	215	216	217
220	221	222	223	224	225	226	227
230	231	232	233	234	235	236	237
240		242	243	244	245	246	
250	251	252	253	254	255	256	257
	261	262	263	264	265	266	267
270	271	272	273	274	275	276	
300	301	302	303	304	305	306	307
310	311	312	313	314	315	316	317
320		322	323	324	325	326	327
330	331	332	333	334	335	336	337
340	341	342	343	344	345	346	
350	351	352	353	354	355	356	357
360		362	363	364	365	366	367
370	371	372	373	374	375	376	377

ISO 8859-1 to ISO 646sv (SWEDISH)

For the conversion of ISO 8859-1 to ISO 646sv all characters not in the following tables are mapped unchanged.

Conversions Performed	
ISO 8859-1	ISO 646sv
304	133
305	135
311	100
326	134
334	136
344	173
345	175
351	140
366	174
374	176

Converted to Underscore '_' (137)

100	133	134	135	136	140		
173	174	175	176				
200	201	202	203	204	205	206	207
210	211	212	213	214	215	216	217
220	221	222	223	224	225	226	227
230	231	232	233	234	235	236	237
240	241	242	243	244	245	246	247
250	251	252	253	254	255	256	257
260	261	262	263	264	265	266	267
270	271	272	273	274	275	276	277
300	301	302	303			306	307
310		312	313	314	315	316	317
320	321	322	323	324	325		327
330	331	332	333		335	336	337
340	341	342	343			346	347
350		352	353	354	355	356	357
360	361	362	363	364	365		367
370	371	372	373		375	376	377

FILES

/usr/lib/iconv/iconv_data	lists the conversions supported
/usr/lib/iconv/*	conversion tables
/usr/lib/locale/*locale***/LC_MESSAGES/uxmesg**	language-specific message file [See **LANG** on **environ**(5).]

SEE ALSO

iconv(1)

NAME

`langinfo` – language information constants

SYNOPSIS

```
#include <langinfo.h>
```

DESCRIPTION

This header file contains the constants used to identify items of **`langinfo`** data. The mode of *items* is given in `nl_types`(5).

`DAY_1`	Locale's equivalent of ''sunday''
`DAY_2`	Locale's equivalent of ''monday''
`DAY_3`	Locale's equivalent of ''tuesday''
`DAY_4`	Locale's equivalent of ''wednesday''
`DAY_5`	Locale's equivalent of ''thursday''
`DAY_6`	Locale's equivalent of ''friday''
`DAY_7`	Locale's equivalent of ''saturday''
`ABDAY_1`	Locale's equivalent of ''sun''
`ABDAY_2`	Locale's equivalent of ''mon''
`ABDAY_3`	Locale's equivalent of ''tue''
`ABDAY_4`	Locale's equivalent of ''wed''
`ABDAY_5`	Locale's equivalent of ''thur''
`ABDAY_6`	Locale's equivalent of ''fri''
`ABDAY_7`	Locale's equivalent of ''sat''
`MON_1`	Locale's equivalent of ''january''
`MON_2`	Locale's equivalent of ''february''
`MON_3`	Locale's equivalent of ''march''
`MON_4`	Locale's equivalent of ''april''
`MON_5`	Locale's equivalent of ''may''
`MON_6`	Locale's equivalent of ''june''
`MON_7`	Locale's equivalent of ''july''
`MON_8`	Locale's equivalent of ''august''
`MON_9`	Locale's equivalent of ''september''
`MON_10`	Locale's equivalent of ''october''
`MON_11`	Locale's equivalent of ''november''
`MON_12`	Locale's equivalent of ''december''
`ABMON_1`	Locale's equivalent of ''jan''

ABMON_2	Locale's equivalent of ''feb''
ABMON_3	Locale's equivalent of ''mar''
ABMON_4	Locale's equivalent of ''apr''
ABMON_5	Locale's equivalent of ''may''
ABMON_6	Locale's equivalent of ''jun''
ABMON_7	Locale's equivalent of ''jul''
ABMON_8	Locale's equivalent of ''aug''
ABMON_9	Locale's equivalent of ''sep''
ABMON_10	Locale's equivalent of ''oct''
ABMON_11	Locale's equivalent of ''nov''
ABMON_12	Locale's equivalent of ''dec''
RADIXCHAR	Locale's equivalent of ''.''
THOUSEP	Locale's equivalent of '',''
YESSTR	Locale's equivalent of ''yes''
NOSTR	Locale's equivalent of ''no''
CRNCYSTR	Locale's currency symbol
D_T_FMT	Locale's default format for date and time
D_FMT	Locale's default format for the date
T_FMT	Locale's default format for the time
AM_STR	Locale's equivalent of ''AM''
PM_STR	Locale's equivalent of ''PM''

This information is retrieved by **nl_langinfo**(3C).

The items **CRNCYSTR**, **RADIXCHAR**, and **THOUSEP** are extracted from the fields **currency_symbol**, **decimal_point**, and **thousands_sep** in the structure returned by **localeconv**(3C).

The items **T_FMT**, **D_FMT**, **D_T_FMT**, **YESSTR**, and **NOSTR** are retrieved from a special message catalog named **Xopen_info** which should be generated for each locale supported and installed in the appropriate directory [see **mkmsgs**(1) and **gettxt**(3C)]. This catalog should have the messages in the order **T_FMT**, **D_FMT**, **D_T_FMT**, **YESSTR**, and **NOSTR**.

All other items are as returned by **strftime**(3C).

SEE ALSO

cftime(3C), **gettxt**(3C), **localeconv**(3C), **mkmsgs**(1), **nl_langinfo**(3C), **nl_types**(5), **strftime**(3C)

NAME

man – macros to format Reference Manual pages

SYNOPSIS

nroff -man *filename*...

troff -man *filename*...

DESCRIPTION

These macros are used to lay out the reference pages in this manual. Note: if *filename* contains format input for a preprocessor, the commands shown above must be piped through the appropriate preprocessor. This is handled automatically by **man**(1). See the "Conventions" section.

Any text argument *t* may be zero to six words. Quotes may be used to include SPACE characters in a word. If *text* is empty, the special treatment is applied to the next input line with text to be printed. In this way **.I** may be used to italicize a whole line, or **.SB** may be used to make small bold letters.

A prevailing indent distance is remembered between successive indented paragraphs, and is reset to default value upon reaching a non-indented paragraph. Default units for indents *i* are ens.

Type font and size are reset to default values before each paragraph, and after processing font and size setting macros.

These strings are predefined by **-man**:

*R	'®', '(Reg)' in **nroff**.
*S	Change to default type size.

Requests

* n.t.l. = next text line; p.i. = prevailing indent

Request	Cause Break	If no Argument	Explanation
.B *t*	no	*t*=n.t.l.*	Text is in bold font.
.BI *t*	no	*t*=n.t.l.	Join words, alternating bold and italic.
.BR *t*	no	*t*=n.t.l.	Join words, alternating bold and roman.
.DT	no	.5i 1i . . .	Restore default tabs.
.HP *i*	yes	*i*=p.i.*	Begin paragraph with hanging indent. Set prevailing indent to *i*.
.I *t*	no	*t*=n.t.l.	Text is italic.
.IB *t*	no	*t*=n.t.l.	Join words, alternating italic and bold.
.IP *x i*	yes	*x*=""	Same as **.TP** with tag *x*.
.IR *t*	no	*t*=n.t.l.	Join words, alternating italic and roman.
.IX *t*	no	-	Index macro.
.LP	yes	-	Begin left-aligned paragraph. Set prevailing indent to .5i.
.PD *d*	no	*d*=.4v	Set vertical distance between paragraphs.
.PP	yes	-	Same as **.LP**.

Request	Cause Break	If no Argument	Explanation
.RE	yes	-	End of relative indent. Restores prevailing indent.
.RB *t*	no	*t*=n.t.l.	Join words, alternating roman and bold.
.RI *t*	no	*t*=n.t.l.	Join words, alternating roman and italic.
.RS *i*	yes	*i*=p.i.	Start relative indent, increase indent by *i*. Sets prevailing indent to .5i for nested indents.
.SB *t*	no	-	Reduce size of text by 1 point, make text bold.
.SH *t*	yes	-	Section Heading.
.SM *t*	no	*t*=n.t.l.	Reduce size of text by 1 point.
.SS *t*	yes	*t*=n.t.l.	Section Subheading.
.TH *n s d f m*	yes	-	Begin reference page *n*, of of section *s*; *d* is the date of the most recent change. If present, *f* is the left page footer; *m* is the main page (center) header. Sets prevailing indent and tabs to .5i.
.TP *i*	yes	*i*=p.i.	Begin indented paragraph, with the tag given on the next text line. Set prevailing indent to *i*.

Conventions

When formatting a manual page, **man** examines the first line to determine whether it requires special processing. For example a first line consisting of:

```
'\" t
```

indicates that the manual page must be run through the **tbl**(1) preprocessor.

A typical manual page for a command or function is laid out as follows:

.TH *title* [1-8]

The name of the command or function, which serves as the title of the manual page. This is followed by the number of the section in which it appears.

.SH NAME

The name, or list of names, by which the command is called, followed by a dash and then a one-line summary of the action performed. All in roman font, this section contains no **troff**(1) commands or escapes, and no macro requests. It is used to generate the **whatis**(1) database.

.SH SYNOPSIS

Commands:

The syntax of the command and its arguments, as typed on the command line. When in boldface, a word must be typed exactly as printed. When in italics, a word can be replaced with an argument that you supply. References to bold or italicized items are not capitalized in other sections, even when they begin a sentence.

Syntactic symbols appear in roman face:

[ ]	An argument, when surrounded by brackets is optional.
|	Arguments separated by a vertical bar are exclusive. You can supply only one item from such a list.
. . .	Arguments followed by an ellipsis can be repeated. When an ellipsis follows a bracketed set, the expression within the brackets can be repeated.

Functions:

If required, the data declaration, or **`#include`** directive, is shown first, followed by the function declaration. Otherwise, the function declaration is shown.

`.SH DESCRIPTION`

A narrative overview of the command or function's external behavior. This includes how it interacts with files or data, and how it handles the standard input, standard output and standard error. Internals and implementation details are normally omitted. This section attempts to provide a succinct overview in answer to the question, "what does it do?"

Literal text from the synopsis appears in constant width, as do literal filenames and references to items that appear elsewhere in the reference manuals. Arguments are italicized.

If a command interprets either subcommands or an input grammar, its command interface or input grammar is normally described in a USAGE section, which follows the OPTIONS section. The DESCRIPTION section only describes the behavior of the command, not that of subcommands.

`.SH OPTIONS`

The list of options along with a description of how each affects the command's operation.

`.SH FILES`

A list of files associated with the command or function.

`.SH SEE ALSO`

A comma-separated list of related manual pages, followed by references to other published materials.

`.SH DIAGNOSTICS`

A list of diagnostic messages and an explanation of each.

`.SH NOTES`

A description of limitations, known defects, and possible problems associated with the command or function.

FILES

`/usr/ucblib/doctools/tmac/an`

SEE ALSO

`man`(1), **`nroff`**(1), **`troff`**(1), **`whatis`**(1)

NAME

math – math functions and constants

SYNOPSIS

#include <math.h>

DESCRIPTION

This file contains declarations of all the functions in the Math Library (described in Section 3M), as well as various functions in the C Library (Section 3C) that return floating-point values.

It defines the structure and constants used by the **matherr**(3M) error-handling mechanisms, including the following constant used as a error-return value:

HUGE	The maximum value of a single-precision floating-point number.

The following mathematical constants are defined for user convenience:

M_E	The base of natural logarithms (*e*).
M_LOG2E	The base-2 logarithm of *e*.
M_LOG10E	The base-10 logarithm of *e*.
M_LN2	The natural logarithm of 2.
M_LN10	The natural logarithm of 10.
M_PI	π, the ratio of the circumference of a circle to its diameter.
M_PI_2	π/2.
M_PI_4	π/4.
M_1_PI	1/π.
M_2_PI	2/π.
M_2_SQRTPI	2/√π.
M_SQRT2	The positive square root of 2.
M_SQRT1_2	The positive square root of 1/2.

The following mathematical constants are also defined in this header file:

MAXFLOAT	The maximum value of a non-infinite single-precision floating point number.
HUGE_VAL	positive infinity.

For the definitions of various machine-dependent constants, see **values**(5).

SEE ALSO

intro(3), **matherr**(3M), **values**(5)

NAME

me – (BSD) macros for formatting papers

SYNOPSIS

nroff -me [**options**] *file* . . .

troff -me [**options**] *file* . . .

DESCRIPTION

This package of **nroff** and **troff** macro definitions provides a canned formatting facility for technical papers in various formats. When producing 2-column output on a terminal, filter the output through **col**(1).

The macro requests are defined below. Many **nroff** and **troff** requests are unsafe in conjunction with this package, however, these requests may be used with impunity after the first **.pp**:

.bp	begin new page
.br	break output line here
.sp *n*	insert *n* spacing lines
.ls *n*	(line spacing) *n*=1 single, *n*=2 double space
.na	no alignment of right margin
.ce *n*	center next *n* lines
.ul *n*	underline next *n* lines
.sz +*n*	add *n* to point size

Output of the **eqn, neqn, refer,** and **tbl**(1) preprocessors for equations and tables is acceptable as input.

Requests

In the following list, initialization refers to the first **.pp**, **.lp**, **.ip**, **.np**, **.sh**, or **.uh** macro. This list is incomplete.

Request	Initial Value	Cause Break	Explanation
.(c	-	yes	Begin centered block
.(d	-	no	Begin delayed text
.(f	-	no	Begin footnote
.(l	-	yes	Begin list
.(q	-	yes	Begin major quote
.(x*x*	-	no	Begin indexed item in index *x*
.(z	-	no	Begin floating keep
.)c	-	yes	End centered block
.)d	-	yes	End delayed text
.)f	-	yes	End footnote
.)l	-	yes	End list
.)q	-	yes	End major quote
.)x	-	yes	End index item
.)z	-	yes	End floating keep

Request	Initial Value	Cause Break	Explanation
.++ *m H*	-	no	Define paper section. *m* defines the part of the paper, and can be **C** (chapter), **A** (appendix), **P** (preliminary, for instance, abstract, table of contents, and so on), **B** (bibliography), **RC** (chapters renumbered from page one each chapter), or **RA** (appendix renumbered from page one).
.+c *T*	-	yes	Begin chapter (or appendix, and so on, as set by **.++**). *T* is the chapter title.
.1c	1	yes	One column format on a new page.
.2c	1	yes	Two column format.
.EN	-	yes	Space after equation produced by **eqn** or **meqn**.
.EQ *x y*	-	yes	Precede equation; break out and add space. Equation number is *y*. The optional argument *x* may be *I* to indent equation (default), *L* to left-adjust the equation, or *C* to center the equation.
.TE	-	yes	End table.
.TH	-	yes	End heading section of table.
.TS *x*	-	yes	Begin table; if *x* is *H* table has repeated heading.
.ac *A N*	-	no	Set up for ACM style output. *A* is the Author's name(s), *N* is the total number of pages. Must be given before the first initialization.
.b *x*	no	no	Print *x* in boldface; if no argument switch to boldface.
.ba *+n*	0	yes	Augments the base indent by *n*. This indent is used to set the indent on regular text (like paragraphs).
.bc	no	yes	Begin new column
.bi *x*	no	no	Print *x* in bold italics (nofill only)
.bx *x*	no	no	Print *x* in a box (nofill only).
.ef *'x'y'z*	'''''	no	Set even footer to *x y z*
.eh *'x'y'z*	'''''	no	Set even header to *x y z*
.fo *'x'y'z*	'''''	no	Set footer to *x y z*
.hx	-	no	Suppress headers and footers on next page.
.he *'x'y'z*	'''''	no	Set header to x y z
.hl	-	yes	Draw a horizontal line
.i *x*	no	no	Italicize *x*; if *x* missing, italic text follows.

Request	Initial Value	Cause Break	Explanation
.ip *x y*	no	yes	Start indented paragraph, with hanging tag *x*. Indentation is *y* ens (default 5).
.lp	yes	yes	Start left-blocked paragraph.
.lo	-	no	Read in a file of local macros of the form **.***x. Must be given before initialization.
.np	1	yes	Start numbered paragraph.
.of *´x´y´z*	´´´´´	no	Set odd footer to x y z
.oh *´x´y´z*	´´´´´	no	Set odd header to x y z
.pd	-	yes	Print delayed text.
.pp	no	yes	Begin paragraph. First line indented.
.r	yes	no	Roman text follows.
.re	-	no	Reset tabs to default values.
.sc	no	no	Read in a file of special characters and diacritical marks. Must be given before initialization.
.sh *n x*	-	yes	Section head follows, font automatically bold. *n* is level of section, *x* is title of section.
.sk	no	no	Leave the next page blank. Only one page is remembered ahead.
.sz +*n*	10p	no	Augment the point size by *n* points.
.th	no	no	Produce the paper in thesis format. Must be given before initialization.
.tp	no	yes	Begin title page.
.u *x*	-	no	Underline argument (even in **troff**). (Nofill only).
.uh	-	yes	Like .sh but unnumbered.
.xp *x*	-	no	Print index *x*.

FILES

/usr/ucblib/doctools/tmac/e
/usr/ucblib/doctools/tmac/*.me

SEE ALSO

col(1), **eqn**(1), **nroff**(1), **troff**(1), **refer**(1), **tbl**(1)

NAME

`ms` – (BSD) text formatting macros

SYNOPSIS

`nroff -ms` [*options*] *file* . . .

`troff -ms` [*options*] *file* . . .

DESCRIPTION

This package of `nroff`(1) and `troff`(1) macro definitions provides a formatting facility for various styles of articles, theses, and books. When producing 2-column output on a terminal or lineprinter, or when reverse line motions are needed, filter the output through `col`(1). All external `-ms` macros are defined below.

Many `nroff` and `troff` requests are unsafe in conjunction with this package. However, the first four requests below may be used with impunity after initialization, and the last two may be used even before initialization:

`.bp`	begin new page
`.br`	break output line
`.sp` *n*	insert *n* spacing lines
`.ce` *n*	center next *n* lines
`.ls` *n*	line spacing: *n*=`1` single, *n*=`2` double space
`.na`	no alignment of right margin

Font and point size changes with `\f` and `\s` are also allowed; for example, `\fIword\fR` will italicize *word*. Output of the `tbl`(1), `eqn`(1) and `refer`(1) preprocessors for equations, tables, and references is acceptable as input.

Requests

Macro Name	Initial Value	Break? Reset?	Explanation
`.AB` *x*	–	y	begin abstract; if *x*=no do not label abstract
`.AE`	–	y	end abstract
`.AI`	–	y	author's institution
`.AM`	–	n	better accent mark definitions
`.AU`	–	y	author's name
`.B` *x*	–	n	embolden *x*; if no *x*, switch to boldface
`.B1`	–	y	begin text to be enclosed in a box
`.B2`	–	y	end boxed text and print it
`.BD`	–	y	block display; center entire block
`.BT`	date	n	bottom title, printed at foot of page
`.BX` *x*	–	n	print word *x* in a box
`.CD`	–	y	centered display with no keep
`.CM`	if t	n	cut mark between pages
`.CT`	–	y,y	chapter title: page number moved to CF (TM only)
`.DA` *x*	if n	n	force date *x* at bottom of page; today if no *x*
`.DE`	–	y	end display (unfilled text) of any kind
`.DS` *x y*	I	y	begin display with keep; *x*=I, L, C, B; *y*=indent

Macro Name	Initial Value	Break? Reset?	Explanation
.EF *x*	–	n	even page footer *x* (3 part as for **.tl**)
.EH *x*	–	n	even page header *x* (3 part as for **.tl**)
.EN	–	y	end displayed equation produced by **eqn**
.EQ *x y*	–	y	break out equation; *x*=L,I,C; *y*=equation number
.FE	–	n	end footnote to be placed at bottom of page
.FP	–	n	numbered footnote paragraph; may be redefined
.FS *x*	–	n	start footnote; *x* is optional footnote label
.HD	undef	n	optional page header below header margin
.I *x*	–	n	italicize *x*; if no *x*, switch to italics
.ID *y*	8n,.5i	y	indented display with no keep; *y*=indent
.IP *x y*	–	y,y	indented paragraph, with hanging tag *x*; *y*=indent
.IX *x y*	–	y	index words *x y* and so on (up to 5 levels)
.KE	–	n	end keep of any kind
.KF	–	n	begin floating keep; text fills remainder of page
.KS	–	y	begin keep; unit kept together on a single page
.LD	–	y	left display with no keep
.LG	–	n	larger; increase point size by 2
.LP	–	y,y	left (block) paragraph.
.MC *x*	–	y,y	multiple columns; *x*=column width
.ND *x*	if t	n	no date in page footer; *x* is date on cover
.NH *x y*	–	y,y	numbered header; *x*=level, *x*=0 resets, *x*=S sets to *y*
.NL	10p	n	set point size back to normal
.OF *x*	–	n	odd page footer *x* (3 part as for **.tl**)
.OH *x*	–	n	odd page header *x* (3 part as for **.tl**)
.P1	if TM	n	print header on first page
.PP	–	y,y	paragraph with first line indented
.PT	- % -	n	page title, printed at head of page
.PX *x*	–	y	print index (table of contents); *x*=no suppresses title
.QP	–	y,y	quote paragraph (indented and shorter)
.R	on	n	return to Roman font
.RE	5n	y,y	retreat: end level of relative indentation
.RP *x*	–	n	released paper format; *x*=no stops title on first page
.RS	5n	y,y	right shift: start level of relative indentation
.SH	–	y,y	section header, in boldface
.SM	–	n	smaller; decrease point size by 2
.TA	8n,5n	n	set TAB characters to 8n 16n ... (**nroff**) 5n 10n ... (**troff**)
.TC *x*	–	y	print table of contents at end; *x*=no suppresses title
.TE	–	y	end of table processed by **tbl**
.TH	–	y	end multi-page header of table
.TL	–	y	title in boldface and two points larger
.TM	off	n	thesis mode
.TS *x*	–	y,y	begin table; if *x*=H table has multi-page header
.UL *x*	–	n	underline *x*, even in **troff**

Macro Name	Initial Value	Break? Reset?	Explanation
.UX *x*	–	n	UNIX; trademark message first time; *x* appended
.XA *x y*	–	y	another index entry; *x*=page or no for none; *y*=indent
.XE	–	y	end index entry (or series of **.IX** entries)
.XP	–	y,y	paragraph with second and subsequent lines indented
.XS *x y*	–	y	begin index entry; *x*=page or no for none; *y*=indent
.1C	on	y,y	one column format, on a new page
.2C	–	y,y	begin two column format
.]-	–	n	beginning of **refer** reference
.[0	–	n	end of unclassifiable type of reference

Registers

Formatting distances can be controlled in **-ms** by means of built-in number registers. For example, this sets the line length to 6.5 inches:

```
.nr  LL  6.5i
```

Here is a table of number registers and their default values:

Name	Register Controls	Takes Effect	Default
PS	point size	paragraph	10
VS	vertical spacing	paragraph	12
LL	line length	paragraph	6i
LT	title length	next page	same as **LL**
FL	footnote length	next **.FS**	5.5i
PD	paragraph distance	paragraph	1v (if n), .3v (if t)
DD	display distance	displays	1v (if n), .5v (if t)
PI	paragraph indent	paragraph	5n
QI	quote indent	next **.QP**	5n
FI	footnote indent	next **.FS**	2n
PO	page offset	next page	0 (if n), ~1i (if t)
HM	header margin	next page	1i
FM	footer margin	next page	1i
FF	footnote format	next **.FS**	0 (1, 2, 3 available)

When resetting these values, make sure to specify the appropriate units. Setting the line length to 7, for example, will result in output with one character per line. Setting **FF** to 1 suppresses footnote superscripting; setting it to 2 also suppresses indentation of the first line; and setting it to 3 produces an **.IP**-like footnote paragraph.

Here is a list of string registers available in **-ms**; they may be used anywhere in the text:

Name	String's Function
`\*Q`	quote (" in **nroff**, '' in **troff**)
`\*U`	unquote (" in **nroff**, '' in **troff**)
`\*-`	dash (-- in **nroff**, – in **troff**)
`\*(MO`	month (month of the year)
`\*(DY`	day (current date)
`\**`	automatically numbered footnote
`\*´`	acute accent (before letter)
`` \*` ``	grave accent (before letter)
`\*^`	circumflex (before letter)
`\*,`	cedilla (before letter)
`\*:`	umlaut (before letter)
`\*~`	tilde (before letter)

When using the extended accent mark definitions available with **.AM**, these strings should come after, rather than before, the letter to be accented.

FILES

/usr/ucb/lib/doctools/tmac/s
/usr/ucblib/doctools/tmac/ms.???

SEE ALSO

col(1), **eqn**(1), **nroff**(1), **refer**(1), **tbl**(1), **troff**(1)

NOTES

Floating keeps and regular keeps are diverted to the same space, so they cannot be mixed together with predictable results.

NAME

nl_types – native language data types

SYNOPSIS

#include <nl_types.h>

DESCRIPTION

This header file contains the following definitions:

nl_catd	used by the message catalog functions **catopen**, **catgets** and **catclose** to identify a catalog
nl_item	used by **nl_langinfo** to identify items of **langinfo** data. Values for objects of type **nl_item** are defined in **langinfo.h**
NL_SETD	used by **gencat** when no **$set** directive is specified in a message text source file. This constant can be used in subsequent calls to **catgets** as the value of the set identifier parameter.
NL_MGSMAX	maximum number of messages per set
NL_SETMAX	maximum number of sets per catalog
NL_TEXTMAX	maximum size of a message
DEF_NLSPATH	the default search path for locating catalogs

SEE ALSO

catgets(3C), **catopen**(3C), **gencat**(1), **langinfo**(5), **nl_langinfo**(3C)

NAME

priv – include file for user–level privilege definitions

SYNOPSIS

```
#include <priv.h>
```

DESCRIPTION

This header file is used by all user–level privilege commands.

```
#ifndef    _PRIV_H
#define    _PRIV_H

/*****************************************************************
 *
 * Header file used by the user-level privilege commands.
 * Contains several macros used by user-level programs.  The
 * external (user-level) privilege representation consists of:
 *
 *            type | privilege
 *
 * and is defined in <sys/privilege.h>.
 *
 * Also contained in <sys/privilege.h> are the definitions
 * for the privilege position names used by user-level macros
 * pm_work(), pm_max(), pm_fixed(), and pm_inher() below.
 *
 * The syntax for privileges are as follows:
 *
 *                          <pname> ::= <A-Z><0-9>
 *              privilege position ::= P_<pname>
 *         string name of privilege ::= lowercase(<pname>)
 *
 * It also contains macro definitions for the command arguments
 * to the filepriv(), procpriv(), procprivl(), and procprivc()
 * calls in addition to the typedef for the user-level definition
 * of a privilege type and privilege set.
 *
 *****************************************************************/

#include   <sys/privilege.h>

/*****************************************************************
 *
 * The following macros are used to specify which privilege sets
 * are updated with the specified privilege.
 *
 *****************************************************************/

#define    pm_work(p)      (priv_t)(((p) & PS_TYPE) ? -1 : ((p) | PS_WKG))
#define    pm_max(p)       (priv_t)(((p) & PS_TYPE) ? -1 : ((p) | PS_MAX))
#define    pm_fixed(p)     (priv_t)(((p) & PS_TYPE) ? -1 : ((p) | PS_FIX))
```

```
#define    pm_inher(p)      (priv_t)(((p) & PS_TYPE) ? -1 : ((p) | PS_INH))

/*****************************************************************
 *
 * The following macros are used to simplify the procprivl(3C) and
 * the procprivc(3C) calls for setting/clearing process privileges.
 *
 *****************************************************************/

#define    OWNER_W          pm_work(P_OWNER)
#define    AUDIT_W          pm_work(P_AUDIT)
#define    COMPAT_W         pm_work(P_COMPAT)
#define    DACREAD_W        pm_work(P_DACREAD)
#define    DACWRITE_W       pm_work(P_DACWRITE)
#define    DEV_W            pm_work(P_DEV)
#define    FILESYS_W        pm_work(P_FILESYS)
#define    MACREAD_W        pm_work(P_MACREAD)
#define    MACWRITE_W       pm_work(P_MACWRITE)
#define    MOUNT_W          pm_work(P_MOUNT)
#define    MULTIDIR_W       pm_work(P_MULTIDIR)
#define    SETFLEVEL_W      pm_work(P_SETFLEVEL)
#define    SETPLEVEL_W      pm_work(P_SETPLEVEL)
#define    SETSPRIV_W       pm_work(P_SETSPRIV)
#define    SETUID_W         pm_work(P_SETUID)
#define    SYSOPS_W         pm_work(P_SYSOPS)
#define    SETUPRIV_W       pm_work(P_SETUPRIV)
#define    DRIVER_W         pm_work(P_DRIVER)
#define    RTIME_W          pm_work(P_RTIME)
#define    MACUPGRADE_W     pm_work(P_MACUPGRADE)
#define    FSYSRANGE_W      pm_work(P_FSYSRANGE)
#define    AUDITWR_W        pm_work(P_AUDITWR)
#define    TSHAR_W          pm_work(P_TSHAR)
#define    PLOCK_W          pm_work(P_PLOCK)
#define    ALLPRIVS_W       pm_work(P_ALLPRIVS)

#define    READ_W           DACREAD_W,MACREAD_W
#define    WRITE_W          DACWRITE_W,MACWRITE_W
#define    ACCESS_W         READ_W,WRITE_W
#define    PRIVS_W          ACCESS_W,SETSPRIV_W,FSYSRANGE_W

/*****************************************************************
 *
 * The following are the definitions for the privilege functions
 *
 *****************************************************************/

#if defined(__STDC__)

extern     int  filepriv(const char *, int, priv_t *, int);
extern     int   procpriv(int, priv_t *, int);
extern     int   procprivl(int, ...);
extern     int   procprivc(int, ...);

#else

extern     int   filepriv();
extern     int   procpriv();
extern     int   procprivl();
extern     int   procprivc();

#endif     /* __STDC__ */

#endif     /* _PRIV_H */
```

SEE ALSO

filepriv(2), procpriv(2), privilege(5)

NAME

`privilege` – include file for privilege mechanism definitions

SYNOPSIS

`#include <sys/privilege.h>`

DESCRIPTION

This header file is used by all privilege mechanisms. All privileges are defined here, as well as certain operations that are necessary for privilege operations.

```
#ifndef     _ACC_PRIV_PRIVILEGE_H     /* wrapper symbol for kernel use */
#define     _ACC_PRIV_PRIVILEGE_H     /* subject to change without notice */

#ifdef _KERNEL_HEADERS

#ifndef     _UTIL_TYPES_H
#include <util/types.h>         /* REQUIRED */
#endif/* _UTIL_TYPES_H */

#elif defined(_KERNEL)

#include <sys/types.h>          /* REQUIRED */

#endif/* _KERNEL_HEADERS */

/***********************************************************
 *
 * The following is the typedef for the user-level privilege
 * definition.  It is here because kernel routines also need
 * to know about this particular type.
 *
 ***********************************************************/

typedef     unsigned    long  priv_t;

/***********************************************************
 *
 * The following are the known privilege sets.
 *
 *    PS_FIX       for fixed privilege sets
 *    PS_INH       for inheritable privilege sets
 *    PS_MAX       for maximum privilege sets
 *    PS_WKG       for working privilege sets
 *
 ***********************************************************/

#define     PS_FIX      0x66000000
#define     PS_INH      0x69000000
#define     PS_MAX      0x6d000000
#define     PS_WKG      0x77000000
#define     PS_TYPE     0xff000000
```

```
/*********************************************************
 *
 * The following are the supported object types for
 * privilege mechanisms.
 *
 *********************************************************/

#define     PS_FILE_OTYPE     0x0
#define     PS_PROC_OTYPE     0x1

/*********************************************************
 *
 * The following is the set of all known privileges.
 * The define NPRIVS is the number of privileges
 * currently in use.  It should be modified whenever a
 * privilege is added or deleted. Further description
 * of each privilege can be found in intro(2).
 *
 *********************************************************/

#define     NPRIVS      26

#define     P_OWNER        0x00000000
#define     P_AUDIT        0x00000001
#define     P_COMPAT       0x00000002
#define     P_DACREAD      0x00000003
#define     P_DACWRITE     0x00000004
#define     P_DEV          0x00000005
#define     P_FILESYS      0x00000006
#define     P_MACREAD      0x00000007
#define     P_MACWRITE     0x00000008
#define     P_MOUNT        0x00000009
#define     P_MULTIDIR     0x0000000a
#define     P_SETPLEVEL    0x0000000b
#define     P_SETSPRIV     0x0000000c
#define     P_SETUID       0x0000000d
#define     P_SYSOPS       0x0000000e
#define     P_SETUPRIV     0x0000000f
#define     P_DRIVER       0x00000010
#define     P_RTIME        0x00000011
#define     P_MACUPGRADE   0x00000012
#define     P_FSYSRANGE    0x00000013
#define     P_SETFLEVEL    0x00000014
#define     P_AUDITWR      0x00000015
#define     P_TSHAR        0x00000016
#define     P_PLOCK        0x00000017
#define     P_CORE         0x00000018
#define     P_LOADMOD      0x00000019
#define     P_ALLPRIVS     0x00ffffff
```

```
/***********************************************************
 *
 * The following  defines  are recognized by the privilege
 * mechanisms.  They are returned in the argument value of
 * the secsys() system call in the form of flags when  the
 * command is ES_PRVINFO.
 *
 ***********************************************************/

#define      PM_UIDBASE      0x00000001
#define      PM_ULVLINIT     0x00000002
#define      PM_PRVMODE      0x00000004

/***********************************************************
 *
 * The following are the CMDS recognized by the procpriv()
 * and filepriv() system calls.
 *
 ***********************************************************/

#define      SETPRV          0x0
#define      CLRPRV          0x1
#define      PUTPRV          0x2
#define      GETPRV          0x3
#define      CNTPRV          0x4

/***********************************************************
 *
 * Structure definition for the privilege sets supported
 * by individual privilege servers.  Also some defines
 * that are used at user-level related to the privilege
 * mechanisms.
 *
 ***********************************************************/

#define      PRVNAMSIZ    32
#define      PRVMAXSETS  256

typedef      struct      pm_setdef {
      priv_t      sd_mask;           /* masked type for this privilege set */
      uint        sd_setcnt;         /* number of privileges in this set   */
      char        sd_name[PRVNAMSIZ]; /* name of this privilege set        */
      ulong_t     sd_objtype;        /* object type of this privilege set  */
} setdef_t;

#if defined(_KERNEL) || defined(_KMEMUSER)

/***********************************************************
 *
 * The following macros are used by the different privilege
 * servers to manipulate privilege bits.
 *
 ***********************************************************/
```

```
/* Turn on significant bits within kernel privilege vector. */
#define     pm_allon          ((1 << NPRIVS) - 1)

/* Mask off type field within privilege descriptor and returns */
/* the privilege */
#define     pm_pos(p)         (pvec_t)((p) & ~PS_TYPE)

/* Mask off the privilege field and return the privilege type */
#define     pm_type(p)        (pvec_t)((p) & PS_TYPE)

/* Convert privilege type to ASCII character */
#define     pm_pridc(p)       (pvec_t)((p) >> 24)

/* Set the pvec_t bit corresponding to the privilege passed */
#define     pm_privbit(p)           (pvec_t)(1 << (p))

/* Convert an ASCII character to a privilege type */
#define     pm_pridt(p)       (pvec_t)((p) << 24)

/* Validate the privilege descripter passed */
#define     pm_invalid(p)           (((pm_pos((p)) < 0 || pm_pos((p))
                             > NPRIVS) && pm_pos((p)) != P_ALLPRIVS) ? 1 : 0)

/* Turn on privilege passed within vector passed */
#define     pm_setbits(p, v)  (v |= (((p) == P_ALLPRIVS)
                            ? pm_allon : (1<<pm_pos(p))))

/*
 * Check the credential(a) passed,
 * to determine if privilege(b) is on within the working privilege set
 */
#define pm_privon(a, b)       ((a)->cr_wkgpriv & (b))

/*
 * Check both credentials(a,b) passed,
 * to determine if the maximum privilege set of (b) is a subset of (a)
 */
#define     pm_subset(a, b)         (((a)->cr_maxpriv & (b)->cr_maxpriv) ==
                                (b)->cr_maxpriv)

/*
 * If the maximum privileges in the credentials passed are non-zero
 * then the process is privileged.
 */
#define     pm_privileged(a)  ((a)->cr_maxpriv)

/************************************************************
 *
 * Structure definitions for the kernel privilege table
 * data types.  Used by any privilege mechanism that stores
 * the information in the kernel.
 *
 ************************************************************/
```

```
/* least privilege file table */
typedef struct    lpftab {
      structlpftab*lpf_next;  /* ptr to next file in list    */
      ino_t lpf_nodeid;       /* node id                     */
      pvec_tlpf_fixpriv;      /* fixed privileges            */
      pvec_tlpf_inhpriv;      /* inheritable privileges      */
      time_tlpf_validity;     /* validity info for integrity */
} lpftab_t;

/* least privilege file system id table */
typedef struct    lpdtab {
      structlpdtab*lpd_next;  /* ptr to next file system in list */
      lpftab_t    *lpd_list;  /* ptr to a privileged file on        */
                              /* this particular file system        */
      dev_t lpd_fsid;         /* the id number for this file system */
} lpdtab_t;

/* least privilege device per file system table */
typedef struct    lpktab {
      structlpktab*lpk_next;  /* ptr to next device in list    */
      lpdtab_t    *lpk_list;  /* ptr to a file system on       */
                              /* this particular device        */
      dev_t lpk_dev;          /* the id number for this device */
} lpktab_t;

#endif/* _KERNEL || _KMEMUSER */

#endif/* _ACC_PRIV_PRIVILEGE_H */
```

SEE ALSO

filepriv(2), procpriv(2), priv(4)

NAME

`prof` – profile within a function

SYNOPSIS

```
#define MARK
#include <prof.h>

void MARK (name);
```

DESCRIPTION

`MARK` introduces a mark called *name* that is treated the same as a function entry point. Execution of the mark adds to a counter for that mark, and program-counter time spent is accounted to the immediately preceding mark or to the function if there are no preceding marks within the active function.

name may be any combination of letters, numbers, or underscores. Each *name* in a single compilation must be unique, but may be the same as any ordinary program symbol.

For marks to be effective, the symbol `MARK` must be defined before the header file `prof.h` is included, either by a preprocessor directive as in the synopsis, or by a command line argument:

```
cc -p -DMARK foo.c
```

If `MARK` is not defined, the `MARK(`*name*`)` statements may be left in the source files containing them and are ignored. `prof -g` must be used to get information on all labels.

EXAMPLE

In this example, marks can be used to determine how much time is spent in each loop. Unless this example is compiled with `MARK` defined on the command line, the marks are ignored.

```
#include <prof.h>
foo( )
{
      int i, j;
      . . .
      MARK(loop1);
      for (i = 0; i < 2000; i++) {
            . . .
      }
      MARK(loop2);
      for (j = 0; j < 2000; j++) {
            . . .
      }
}
```

SEE ALSO

`monitor`(3C) `prof`(1), `profil`(2)

NAME

regexp: **compile**, **step**, **advance** – regular expression compile and match routines

SYNOPSIS

```
#define INIT declarations
#define GETC(void) getc code
#define PEEKC(void) peekc code
#define UNGETC(void) ungetc code
#define RETURN(ptr) return code
#define ERROR(val) error code

#include <regexp.h>

char *compile(char *instring, char *expbuf, char *endbuf, int eof);

int step(char *string, char *expbuf);

int advance(char *string, char *expbuf);

extern char *loc1, *loc2, *locs;
```

DESCRIPTION

These functions are general purpose regular expression matching routines to be used in programs that perform regular expression matching. These functions are defined by the **regexp.h** header file.

The functions **step** and **advance** do pattern matching given a character string and a compiled regular expression as input.

The function **compile** takes as input a regular expression as defined below and produces a compiled expression that can be used with **step** or **advance**.

A regular expression specifies a set of character strings. A member of this set of strings is said to be matched by the regular expression. Some characters have special meaning when used in a regular expression; other characters stand for themselves.

The regular expressions available for use with the regexp functions are constructed as follows:

Expression	*Meaning*
c	the character *c* where *c* is not a special character.
\c	the character *c* where *c* is any character, except a digit in the range `1-9`.
^	the beginning of the line being compared.
$	the end of the line being compared.
.	any character in the input.
`[`*s*`]`	any character in the set *s*, where *s* is a sequence of characters and/or a range of characters, for example, `[`*c*`-`*c*`]`.
`[^`*s*`]`	any character not in the set *s*, where *s* is defined as above.

*r**	zero or more successive occurrences of the regular expression *r*. The longest leftmost match is chosen.
rx	the occurrence of regular expression *r* followed by the occurrence of regular expression *x*. (Concatenation)
r`\{`*m*`,`*n*`\}`	any number of *m* through *n* successive occurrences of the regular expression *r*. The regular expression *r*`\{`*m*`\}` matches exactly *m* occurrences; *r*`\{`*m*`,\}` matches at least *m* occurrences.
`\(`*r*`\)`	the regular expression *r*. When `\`*n* (where *n* is a number greater than zero) appears in a constructed regular expression, it stands for the regular expression *x* where *x* is the *n*th regular expression enclosed in `\(` and `\)` that appeared earlier in the constructed regular expression. For example, `\(`*r*`\)`*x*`\(`*y*`\)`*z*`\2` is the concatenation of regular expressions *rxyzy*.

Characters that have special meaning except when they appear within square brackets (`[ ]`) or are preceded by `\` are: `.`, `*`, `[`, `\`. Other special characters, such as `$` have special meaning in more restricted contexts.

The character `^` at the beginning of an expression permits a successful match only immediately after a newline, and the character `$` at the end of an expression requires a trailing newline.

Two characters have special meaning only when used within square brackets. The character `-` denotes a range, `[`*c*`-`*c*`]`, unless it is just after the open bracket or before the closing bracket, `[-`*c*`]` or `[`*c*`-]` in which case it has no special meaning. When used within brackets, the character `^` has the meaning *complement of* if it immediately follows the open bracket (example: `[^`*c*`]`); elsewhere between brackets (example: `[`*c*`^]`) it stands for the ordinary character `^`.

The special meaning of the `\` operator can be escaped only by preceding it with another `\`, for example, `\\`.

Programs must have the following five macros declared before the **`#include regexp.h`** statement. These macros are used by the **`compile`** routine. The macros **GETC**, **PEEKC**, and **UNGETC** operate on the regular expression given as input to **`compile`**.

GETC	This macro returns the value of the next character (byte) in the regular expression pattern. Successive calls to **GETC** should return successive characters of the regular expression.
PEEKC	This macro returns the next character (byte) in the regular expression. Immediately successive calls to **PEEKC** should return the same character, which should also be the next character returned by **GETC**.
UNGETC	This macro causes the argument **c** to be returned by the next call to **GETC** and **PEEKC**. No more than one character of pushback is ever needed and this character is guaranteed to be the last character read by **GETC**. The return value of the macro **`UNGETC(c)`** is always ignored.

RETURN(*ptr*)	This macro is used on normal exit of the **compile** routine. The value of the argument *ptr* is a pointer to the character after the last character of the compiled regular expression. This is useful to programs which have memory allocation to manage.
ERROR(*val*)	This macro is the abnormal return from the **compile** routine. The argument *val* is an error number [see ERRORS below for meanings]. This call should never return.

The syntax of the **compile** routine is as follows:

> **compile**(*instring*, *expbuf*, *endbuf*, *eof*)

The first parameter, *instring*, is never used explicitly by the **compile** routine but is useful for programs that pass down different pointers to input characters. It is sometimes used in the **INIT** declaration (see below). Programs which call functions to input characters or have characters in an external array can pass down a value of **(char *)0** for this parameter.

The next parameter, *expbuf*, is a character pointer. It points to the place where the compiled regular expression will be placed.

The parameter *endbuf* is one more than the highest address where the compiled regular expression may be placed. If the compiled expression cannot fit in **(endbuf-expbuf)** bytes, a call to **ERROR(50)** is made.

The parameter *eof* is the character which marks the end of the regular expression. This character is usually a **/**.

Each program that includes the **regexp.h** header file must have a **#define** statement for **INIT**. It is used for dependent declarations and initializations. Most often it is used to set a register variable to point to the beginning of the regular expression so that this register variable can be used in the declarations for **GETC**, **PEEKC**, and **UNGETC**. Otherwise it can be used to declare external variables that might be used by **GETC**, **PEEKC** and **UNGETC**. [See EXAMPLE below.]

The first parameter to the **step** and **advance** functions is a pointer to a string of characters to be checked for a match. This string should be null terminated.

The second parameter, *expbuf*, is the compiled regular expression which was obtained by a call to the function **compile**.

The function **step** returns non-zero if some substring of *string* matches the regular expression in *expbuf* and zero if there is no match. If there is a match, two external character pointers are set as a side effect to the call to **step**. The variable **loc1** points to the first character that matched the regular expression; the variable **loc2** points to the character after the last character that matches the regular expression. Thus if the regular expression matches the entire input string, **loc1** will point to the first character of *string* and **loc2** will point to the null at the end of *string*.

The function **advance** returns non-zero if the initial substring of *string* matches the regular expression in *expbuf*. If there is a match, an external character pointer, **loc2**, is set as a side effect. The variable **loc2** points to the next character in *string* after the last character that matched.

When **advance** encounters a `*` or `\{ \}` sequence in the regular expression, it will advance its pointer to the string to be matched as far as possible and will recursively call itself trying to match the rest of the string to the rest of the regular expression. As long as there is no match, **advance** will back up along the string until it finds a match or reaches the point in the string that initially matched the `*` or `\{ \}`. It is sometimes desirable to stop this backing up before the initial point in the string is reached. If the external character pointer **locs** is equal to the point in the string at sometime during the backing up process, **advance** will break out of the loop that backs up and will return zero.

The external variables **circf**, **sed**, and **nbra** are reserved.

DIAGNOSTICS

The function **compile** uses the macro **RETURN** on success and the macro **ERROR** on failure (see above). The functions **step** and **advance** return non-zero on a successful match and zero if there is no match. Errors are:

11	range endpoint too large.
16	bad number.
25	\ *digit* out of range.
36	invalid or missing delimiter.
41	no remembered search string.
42	`\( \)` imbalance.
43	too many `\(`.
44	more than 2 numbers given in `\{ \}`.
45	`}` expected after `\`.
46	first number exceeds second in `\{ \}`.
49	`[ ]` imbalance.
50	regular expression overflow.

EXAMPLES

The following is an example of how the regular expression macros and calls might be defined by an application program:

```
#define INIT       register char *sp = instring;
#define GETC       (*sp++)
#define PEEKC      (*sp)
#define UNGETC(c)  (--sp)
#define RETURN(*c) return;
#define ERROR(c)   regerr

#include <regexp.h>

  . . .
     (void) compile(*argv, expbuf, &expbuf[ESIZE],'\0');
  . . .
     if (step(linebuf, expbuf))
                       succeed;
```

SEE ALSO

regexpr(3G)

NAME

siginfo – signal generation information

SYNOPSIS

#include <siginfo.h>

DESCRIPTION

If a process is catching a signal, it may request information that tells why the system generated that signal [see **sigaction**(2)]. If a process is monitoring its children, it may receive information that tells why a child changed state [see **waitid**(2)]. In either case, the system returns the information in a structure of type **siginfo_t**, which includes the following information:

```
int si_signo    /* signal number */
int si_errno    /* error number */
int si_code     /* signal code */
```

si_signo contains the system-generated signal number. (For the **waitid**(2) function, **si_signo** is always **SIGCHLD**.)

If **si_errno** is non-zero, it contains an error number associated with this signal, as defined in **errno.h**.

si_code contains a code identifying the cause of the signal. If the value of **si_code** is less than or equal to 0, then the signal was generated by a user process [see **kill**(2) and **sigsend**(2)] and the **siginfo** structure contains the following additional information:

```
pid_t si_pid    /* sending process ID */
uid_t si_uid    /* sending user ID */
```

Otherwise, **si_code** contains a signal-specific reason why the signal was generated, as follows:

Signal	Code	Reason
SIGILL	**ILL_ILLOPC**	illegal opcode
	ILL_ILLOPN	illegal operand
	ILL_ILLADR	illegal addressing mode
	ILL_ILLTRP	illegal trap
	ILL_PRVOPC	privileged opcode
	ILL_PRVREG	privileged register
	ILL_COPROC	co-processor error
	ILL_BADSTK	internal stack error
SIGFPE	**FPE_INTDIV**	integer divide by zero
	FPE_INTOVF	integer overflow
	FPE_FLTDIV	floating point divide by zero
	FPE_FLTOVF	floating point overflow
	FPE_FLTUND	floating point underflow
	FPE_FLTRES	floating point inexact result
	FPE_FLTINV	invalid floating point operation
	FPE_FLTSUB	subscript out of range

Signal	Code	Reason
SIGSEGV	**SEGV_MAPERR**	address not mapped to object
	SEGV_ACCERR	invalid permissions for mapped object
SIGBUS	**BUS_ADRALN**	invalid address alignment
	BUS_ADRERR	non-existent physical address
	BUS_OBJERR	object specific hardware error
SIGTRAP	**TRAP_BRKPT**	process breakpoint
	TRAP_TRACE	process trace trap
SIGCHLD	**CLD_EXITED**	child has exited
	CLD_KILLED	child was killed
	CLD_DUMPED	child terminated abnormally
	CLD_TRAPPED	traced child has trapped
	CLD_STOPPED	child has stopped
	CLD_CONTINUED	stopped child had continued
SIGPOLL	**POLL_IN**	data input available
	POLL_OUT	output buffers available
	POLL_MSG	input message available
	POLL_ERR	I/O error
	POLL_PRI	high priority input available
	POLL_HUP	device disconnected

In addition, the following signal-dependent information is available for kernel-generated signals:

Signal	Field	Value
SIGILL **SIGFPE**	**caddr_t si_addr**	address of faulting instruction
SIGSEGV **SIGBUS**	**caddr_t si_addr**	address of faulting memory reference
SIGCHLD	**pid_t si_pid**	child process ID
	int si_status	exit value or signal
SIGPOLL	**long si_band**	band event for **POLL_IN**, **POLL_OUT**, or **POLL_MSG**

SEE ALSO

sigaction(2), **signal**(5), **waitid**(2)

NOTES

For **SIGCHLD** signals, if **si_code** is equal to **CLD_EXITED**, then **si_status** is equal to the exit value of the process; otherwise, it is equal to the signal that caused the process to change state. For some implementations, the exact value of **si_addr** may not be available; in that case, **si_addr** is guaranteed to be on the same page as the faulting instruction or memory reference.

NAME

signal – base signals

SYNOPSIS

#include <signal.h>

DESCRIPTION

A signal is an asynchronous notification of an event. A signal is said to be generated for (or sent to) a process when the event associated with that signal first occurs. Examples of such events include hardware faults, timer expiration and terminal activity, as well as the invocation of the **kill** or **sigsend** system calls. In some circumstances, the same event generates signals for multiple processes. A process may request a detailed notification of the source of the signal and the reason why it was generated [see **siginfo**(5)].

Each process may specify a system action to be taken in response to each signal sent to it, called the signal's disposition. The set of system signal actions for a process is initialized from that of its parent. Once an action is installed for a specific signal, it usually remains installed until another disposition is explicitly requested by a call to either **sigaction, signal** or **sigset**, or until the process execs [see **sigaction**(2) and **signal**(2)]. When a process execs, all signals whose disposition has been set to catch the signal will be set to **SIG_DFL**. Alternatively, a process may request that the system automatically reset the disposition of a signal to **SIG_DFL** after it has been caught [see **sigaction**(2) and **signal**(2)].

A signal is said to be delivered to a process when the appropriate action for the process and signal is taken. During the time between the generation of a signal and its delivery, the signal is said to be pending [see **sigpending**(2)]. Ordinarily, this interval cannot be detected by an application. However, a signal can be blocked from delivery to a process [see **signal**(2) and **sigprocmask**(2)]. If the action associated with a blocked signal is anything other than to ignore the signal, and if that signal is generated for the process, the signal remains pending until either it is unblocked or the signal's disposition requests that the signal be ignored. If the signal disposition of a blocked signal requests that the signal be ignored, and if that signal is generated for the process, the signal is discarded immediately upon generation.

Each process has a signal mask that defines the set of signals currently blocked from delivery to it [see **sigprocmask**(2)]. The signal mask for a process is initialized from that of its parent.

The determination of which action is taken in response to a signal is made at the time the signal is delivered, allowing for any changes since the time of generation. This determination is independent of the means by which the signal was originally generated.

The signals currently defined in **sys/signal.h** are as follows:

Name	Value	Default	Event
SIGHUP	1	Exit	Hangup [see **termio**(7)]
SIGINT	2	Exit	Interrupt [see **termio**(7)]
SIGQUIT	3	Core	Quit [see **termio**(7)]
SIGILL	4	Core	Illegal Instruction
SIGTRAP	5	Core	Trace/Breakpoint Trap
SIGABRT	6	Core	Abort
SIGEMT	7	Core	Emulation Trap
SIGFPE	8	Core	Arithmetic Exception
SIGKILL	9	Exit	Killed
SIGBUS	10	Core	Bus Error
SIGSEGV	11	Core	Segmentation Fault
SIGSYS	12	Core	Bad System Call
SIGPIPE	13	Exit	Broken Pipe
SIGALRM	14	Exit	Alarm Clock
SIGTERM	15	Exit	Terminated
SIGUSR1	16	Exit	User Signal 1
SIGUSR2	17	Exit	User Signal 2
SIGCHLD	18	Ignore	Child Status
SIGPWR	19	Ignore	Power Fail/Restart
SIGWINCH	20	Ignore	Window Size Change
SIGURG	21	Ignore	Urgent Socket Condition
SIGPOLL	22	Ignore	Socket I/O Possible
SIGSTOP	23	Stop	Stopped (signal)
SIGTSTP	24	Stop	Stopped (user) [see **termio**(7)]
SIGCONT	25	Ignore	Continued
SIGTTIN	26	Stop	Stopped (tty input) [see **termio**(7)]
SIGTTOU	27	Stop	Stopped (tty output) [see **termio**(7)]
SIGVTALRM	28	Exit	Virtual Timer Expired
SIGPROF	29	Exit	Profiling Timer Expired
SIGXCPU	30	Core	CPU time limit exceeded [see **getrlimit**(2)]
SIGXFSZ	31	Core	File size limit exceeded [see **getrlimit**(2)]

Using the **signal**, **sigset** or **sigaction** system call, a process may specify one of three dispositions for a signal: take the default action for the signal, ignore the signal, or catch the signal.

Default Action: SIG_DFL

A disposition of **SIG_DFL** specifies the default action. The default action for each signal is listed in the table above and is selected from the following:

Exit When it gets the signal, the receiving process is to be terminated with all the consequences outlined in **exit**(2).

Core When it gets the signal, the receiving process is to be terminated with all the consequences outlined in **exit**(2). In addition, a "core image" of the process is constructed in the current working directory.

Stop When it gets the signal, the receiving process is to stop.

Ignore When it gets the signal, the receiving process is to ignore it. This is identical to setting the disposition to **SIG_IGN**.

Ignore Signal: SIG_IGN

A disposition of **SIG_IGN** specifies that the signal is to be ignored.

Catch Signal: *function address*

A disposition that is a function address specifies that, when it gets the signal, the receiving process is to execute the signal handler at the specified address. Normally, the signal handler is passed the signal number as its only argument; if the disposition was set with the **sigaction** function however, additional arguments may be requested [see **sigaction**(2)]. When the signal handler returns, the receiving process resumes execution at the point it was interrupted, unless the signal handler makes other arrangements. If an invalid function address is specified, results are undefined.

If the disposition has been set with the **sigset** or **sigaction** function, the signal is automatically blocked by the system while the signal catcher is executing. If a **longjmp** [see **setjmp**(3C)] is used to leave the signal catcher, then the signal must be explicitly unblocked by the user [see **signal**(2) and **sigprocmask**(2)].

If execution of the signal handler interrupts a blocked system call, the handler is executed and the interrupted system call returns a –1 to the calling process with **errno** set to **EINTR**. However, if the **SA_RESTART** flag is set the system call will be transparently restarted.

NOTES

The dispositions of the **SIGKILL** and **SIGSTOP** signals cannot be altered from their default values. The system generates an error if this is attempted.

The **SIGKILL** and **SIGSTOP** signals cannot be blocked. The system silently enforces this restriction.

Whenever a process receives a **SIGSTOP**, **SIGTSTP**, **SIGTTIN**, or **SIGTTOU** signal, regardless of its disposition, any pending **SIGCONT** signal is discarded. A process stopped by the above four signals is said to be in a job control stop.

Whenever a process receives a **SIGCONT** signal, regardless of its disposition, any pending **SIGSTOP**, **SIGTSTP**, **SIGTTIN**, and **SIGTTOU** signals are discarded. In addition, if the process was stopped, it is continued.

SIGPOLL is issued when a file descriptor corresponding to a STREAMS [see **intro**(2)] file has a "selectable" event pending. A process must specifically request that this signal be sent using the **I_SETSIG ioctl** call. Otherwise, the process will never receive **SIGPOLL**.

If the disposition of the **SIGCHLD** signal has been set with **signal** or **sigset**, or with **sigaction** and the **SA_NOCLDSTOP** flag has been specified, it will only be sent to the calling process when its children exit; otherwise, it will also be sent when the calling process's children are stopped or continued due to job control.

For backward compatibility, the names **SIGCLD**, **SIGIOT**, and **SIGIO** are defined in this header file. **SIGCLD** identifies the same signal as **SIGCHLD**. **SIGIOT** identifies the same signal as **SIGABRT**, and **SIGIO** identifies the same signal as **SIGPOLL**. However, new applications should use **SIGCHLD**, **SIGABRT**, and **SIGPOLL**.

The disposition of signals that are inherited as **SIG_IGN** should not be changed.

SEE ALSO

exit(2), **getrlimit**(2), **intro**(2), **kill**(2), **pause**(2), **sigaction**(2), **sigaltstack**(2), **siginfo**(5), **signal**(2), **sigprocmask**(2), **sigsend**(2), **sigsetops**(3C), **sigsuspend**(2), **ucontext**(5), **wait**(2)

NAME

stat – data returned by stat system call

SYNOPSIS

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

DESCRIPTION

The system calls **stat**, **lstat** and **fstat** return data in a **stat** structure, which is defined in **stat.h** and includes the following members:

```
dev_t                 st_dev;
ino_t                 st_ino;
mode_t                st_mode;
nlink_t               st_nlink;
uid_t                 st_uid;
gid_t                 st_gid;
dev_t                 st_rdev;
off_t                 st_size;
time_t                st_atime;
time_t                st_mtime;
time_t                st_ctime;
long                  st_blksize;
long                  st_blocks;
char                  st_fstype[_ST_FSTYPSZ];
int                   st_aclcnt;
level_t st_level;
ulong_t st_flags;     /* general purpose flag */
```

The following members are only valid if Enhanced Security is installed:

```
int st_aclcnt;
level_t st_level;
ulong_t st_flags;
```

The constants used in the **st_mode** field are also defined in this file:

```
#define   S_IFMT     /* type of file */
#define   S_IAMB     /* access mode bits */
#define   S_IFIFO    /* fifo */
#define   S_IFCHR    /* character special */
#define   S_IFDIR    /* directory */
#define   S_IFNAM    /* XENIX special named file */
#define   S_INSEM    /* XENIX semaphore subtype of IFNAM */
#define   S_INSHD    /* XENIX shared data subtype of IFNAM */
#define   S_IFBLK    /* block special */
#define   S_IFREG    /* regular */
#define   S_IFLNK    /* symbolic link */
#define   S_ISUID    /* set user id on execution */
#define   S_ISGID    /* set group id on execution */
#define   S_ISVTX    /* save swapped text even after use */
```

```
        #define   S_IREAD     /* read permission, owner */
        #define   S_IWRITE    /* write permission, owner */
        #define   S_IEXEC     /* execute/search permission, owner */
        #define   S_ENFMT     /* record locking enforcement flag */
        #define   S_IRWXU     /* read, write, execute: owner */
        #define   S_IRUSR     /* read permission: owner */
        #define   S_IWUSR     /* write permission: owner */
        #define   S_IXUSR     /* execute permission: owner */
        #define   S_IRWXG     /* read, write, execute: group */
        #define   S_IRGRP     /* read permission: group */
        #define   S_IWGRP     /* write permission: group */
        #define   S_IXGRP     /* execute permission: group */
        #define   S_IRWXO     /* read, write, execute: other */
        #define   S_IROTH     /* read permission: other */
        #define   S_IWOTH     /* write permission: other */
        #define   S_IXOTH     /* execute permission: other */
```

The following macros are for POSIX conformance:

```
        #define   S_ISBLK(mode)      block special file
        #define   S_ISCHR(mode)      character special file
        #define   S_ISDIR(mode)      directory file
        #define   S_ISFIFO(mode)     pipe or fifo file
        #define   S_ISREG(mode)      regular file
```

One constant used in the **st_flags** field is also defined in this file:

```
        #define   S_ISMLD   /* indicates multilevel directory */
```

Multilevel directories are only supported if the Enhanced Security Utilities are installed.

SEE ALSO

stat(2), **types**(5)

NAME

stdarg – handle variable argument list

SYNOPSIS

#include <stdarg.h>

va_list *pvar***;**

void va_start(va_list *pvar***,** *parmN***);**

type **va_arg(va_list** *pvar***,** *type***);**

void va_end(va_list *pvar***);**

DESCRIPTION

This set of macros allows portable procedures that accept variable numbers of arguments of variable types to be written. Routines that have variable argument lists (such as **printf**) but do not use **stdarg** are inherently non-portable, as different machines use different argument-passing conventions.

va_list is a type defined for the variable used to traverse the list.

The **va_start** macro is invoked before any access to the unnamed arguments and initializes *pvar* for subsequent use by **va_arg** and **va_end**. The parameter *parmN* is the identifier of the rightmost parameter in the variable parameter list in the function definition (the one just before the **, ...**). If this parameter is declared with the **register** storage class or with a function or array type, or with a type that is not compatible with the type that results after application of the default argument promotions, the behavior is undefined.

The parameter *parmN* is required under strict ANSI C compilation. In other compilation modes, *parmN* need not be supplied and the second parameter to the **va_start** macro can be left empty [e.g., **va_start(***pvar***,);**]. This allows for routines that contain no parameters before the **...** in the variable parameter list.

The **va_arg** macro expands to an expression that has the type and value of the next argument in the call. The parameter *pvar* should have been previously initialized by **va_start**. Each invocation of **va_arg** modifies *pvar* so that the values of successive arguments are returned in turn. The parameter *type* is the type name of the next argument to be returned. The type name must be specified in such a way so that the type of a pointer to an object that has the specified type can be obtained simply by postfixing a ***** to *type*. If there is no actual next argument, or if *type* is not compatible with the type of the actual next argument (as promoted according to the default argument promotions), the behavior is undefined.

The **va_end** macro is used to clean up.

Multiple traversals, each bracketed by **va_start** and **va_end**, are possible.

EXAMPLE

This example gathers into an array a list of arguments that are pointers to strings (but not more than **MAXARGS** arguments) with function **f1**, then passes the array as a single argument to function **f2**. The number of pointers is specified by the first argument to **f1**.

```
#include <stdarg.h>
#define MAXARGS 31

void f1(int n_ptrs, ...)
{
     va_list ap;
     char *array[MAXARGS];
     int ptr_no = 0;

     if (n_ptrs > MAXARGS)
          n_ptrs = MAXARGS;
     va_start(ap, n_ptrs);
     while (ptr_no < n_ptrs)
          array[ptr_no++] = va_arg(ap, char*);
     va_end(ap);
     f2(n_ptrs, array);
}
```

Each call to `f1` shall have visible the definition of the function or a declaration such as

```
void f1(int, ...)
```

SEE ALSO

`vprintf`(3S)

NOTES

It is up to the calling routine to specify in some manner how many arguments there are, since it is not always possible to determine the number of arguments from the stack frame. For example, **`execl`** is passed a zero pointer to signal the end of the list. **`printf`** can tell how many arguments there are by the format. It is non-portable to specify a second argument of **`char`**, **`short`**, or **`float`** to **`va_arg`**, because arguments seen by the called function are not **`char`**, **`short`**, or **`float`**. C converts **`char`** and **`short`** arguments to **`int`** and converts **`float`** arguments to **`double`** before passing them to a function.

NAME

term – conventional names for terminals

DESCRIPTION

Terminal names are maintained as part of the shell environment in the environment variable **TERM** [see **sh**(1), **profile**(4), and **environ**(5)]. These names are used by certain commands [for example, **tabs**, **tput**, and **vi**] and certain functions [for example, see **curses**(3curses)].

Files under **/usr/share/lib/terminfo** are used to name terminals and describe their capabilities. These files are in the format described in **terminfo**(4). Entries in **terminfo** source files consist of a number of comma-separated fields. To print a description of a terminal *term*, use the command **infocmp -I** *term* [see **infocmp**(1M)]. White space after each comma is ignored. The first line of each terminal description in the **terminfo** database gives the names by which **terminfo** knows the terminal, separated by bar (|) characters. The first name given is the most common abbreviation for the terminal [this is the one to use to set the environment variable **TERMINFO** in **$HOME/.profile**; see **profile**(4)], the last name given should be a long name fully identifying the terminal, and all others are understood as synonyms for the terminal name. All names but the last should contain no blanks and must be unique in the first 14 characters; the last name may contain blanks for readability.

Terminal names (except for the last, verbose entry) should be chosen using the following conventions. The particular piece of hardware making up the terminal should have a root name chosen, for example, for the AT&T 4425 terminal, **att4425**. This name should not contain hyphens, except that synonyms may be chosen that do not conflict with other names. Up to 8 characters, chosen from the set **a** through **z** and **0** through **9**, make up a basic terminal name. Names should generally be based on original vendors rather than local distributors. A terminal acquired from one vendor should not have more than one distinct basic name. Terminal sub-models, operational modes that the hardware can be in, or user preferences should be indicated by appending a hyphen and an indicator of the mode. Thus, an AT&T 4425 terminal in 132 column mode is **att4425-w**. The following suffixes should be used where possible:

Suffix	Meaning	Example
-w	Wide mode (more than 80 columns)	**att4425-w**
-am	With auto. margins (usually default)	**vt100-am**
-nam	Without automatic margins	**vt100-nam**
-*n*	Number of lines on the screen	**aaa-60**
-na	No arrow keys (leave them in local)	**c100-na**
-np	Number of pages of memory	**c100-4p**
-rv	Reverse video	**att4415-rv**

To avoid conflicts with the naming conventions used in describing the different modes of a terminal (for example, **-w**), it is recommended that a terminal's root name not contain hyphens. Further, it is good practice to make all terminal names used in the **terminfo**(4) database unique. Terminal entries that are present only for inclusion in other entries via the **use=** facilities should have a '+' in their name, as in **4415+nl**.

Here are some of the known terminal names: (For a complete list, enter the command `ls -C /usr/share/lib/terminfo/?`.)

`2621,hp2621`	Hewlett-Packard 2621 series
`2631`	Hewlett-Packard 2631 line printer
`2631-c`	Hewlett-Packard 2631 line printer, compressed mode
`2631-e`	Hewlett-Packard 2631 line printer, expanded mode
`2640,hp2640`	Hewlett-Packard 2640 series
`2645,hp2645`	Hewlett-Packard 2645 series
`3270`	IBM Model 3270
`33,tty33`	AT&T Teletype Model 33 KSR
`35,tty35`	AT&T Teletype Model 35 KSR
`37,tty37`	AT&T Teletype Model 37 KSR
`4000a`	Trendata 4000a
`4014,tek4014`	TEKTRONIX 4014
`40,tty40`	AT&T Teletype Dataspeed 40/2
`43,tty43`	AT&T Teletype Model 43 KSR
`4410,5410`	AT&T 4410/5410 in 80-column mode, version 2
`4410-nfk,5410-nfk`	AT&T 4410/5410 without function keys, version 1
`4410-nsl,5410-nsl`	AT&T 4410/5410 without pln defined
`4410-w,5410-w`	AT&T 4410/5410 in 132-column mode
`4410v1,5410v1`	AT&T 4410/5410 in 80-column mode, version 1
`4410v1-w,5410v1-w`	AT&T 4410/5410 in 132-column mode, version 1
`4418,5418`	AT&T 5418 in 80-column mode
`4418-w,5418-w`	AT&T 5418 in 132-column mode
`4420`	AT&T Teletype Model 4420
`4424`	AT&T Teletype Model 4424
`4424-2`	AT&T Teletype Model 4424 in display function group ii
`4425,5425`	AT&T 4425/5425
`4425-fk,5425-fk`	AT&T 4425/5425 without function keys
`4425-nl,5425-nl`	AT&T 4425/5425 without changing labels in 80-column mode
`4425-w,5425-w`	AT&T 4425/5425 in 132-column mode
`4425-w-fk,5425-w-fk`	AT&T 4425/5425 without function keys in 132-column mode
`4425-nl-w,5425-nl-w`	AT&T 4425/5425 without changing labels in 132-column mode
`4426`	AT&T Teletype Model 4426S

450	DASI 450 (same as Diablo 1620)
450-12	DASI 450 in 12-pitch mode
500,att500	AT&T-IS 500 terminal
510,510a	AT&T 510/510a in 80-column mode
513bct,att513	AT&T 513 bct terminal
5320	AT&T 5320 hardcopy terminal
5420_2	AT&T 5420 model 2 in 80-column mode
5420_2-w	AT&T 5420 model 2 in 132-column mode
610,610bct	AT&T 610 bct terminal in 80-column mode
610-w,610bct-w	AT&T 610 bct terminal in 132-column mode
630,630MTG	AT&T 630 Multi-Tasking Graphics terminal
735,ti	Texas Instruments TI735 and TI725
745	Texas Instruments TI745
dumb	generic name for terminals that lack reverse line-feed and other special escape sequences
hp	Hewlett-Packard (same as 2645)
lp	generic name for a line printer
pt505	AT&T Personal Terminal 505 (22 lines)
pt505-24	AT&T Personal Terminal 505 (24-line mode)
sync	generic name for synchronous Teletype Model 4540-compatible terminals

Commands whose behavior depends on the type of terminal should accept arguments of the form **-T***term* where *term* is one of the names given above; if no such argument is present, such commands should obtain the terminal type from the environment variable **TERM**, which, in turn, should contain *term*.

FILES

/usr/share/lib/terminfo/?/* compiled terminal description database

SEE ALSO

curses(3curses), **environ**(5), **infocmp**(1M), **profile**(4), **sh**(1), **stty**(1), **tabs**(1), **terminfo**(4), **tput**(1), **vi**(1)

NAME

types – primitive system data types

SYNOPSIS

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

DESCRIPTION

The data types defined in **types.h** are used in UNIX system code. Some data of these types are accessible to user code:

```
typedef  struct { int r[1]; } *physadr;
typedef  long                clock_t;
typedef  long                daddr_t;
typedef  char *              caddr_t;
typedef  unsigned char       unchar;
typedef  unsigned short      ushort;
typedef  unsigned int        uint;
typedef  unsigned long       ulong;
typedef  unsigned long       ino_t;
typedef  long                uid_t;
typedef  long                gid_t;
typedef  ulong               nlink_t;
typedef  ulong               mode_t;
typedef  short               cnt_t;
typedef  long                time_t;
typedef  int                 label_t[10];
typedef  ulong               dev_t;
typedef  long                off_t;
typedef  long                pid_t;
typedef  ulong               paddr_t;
typedef  int                 key_t;
typedef  unsigned char       use_t;
typedef  short               sysid_t;
typedef  short               index_t;
typedef  short               lock_t;
typedef  unsigned int        size_t;
typedef  long                clock_t;
typedef  long                pid_t;
typedef  int                 ssize_t;
```

The form **daddr_t** is used for disk addresses except in an i-node on disk, see **fs**(4). Times are encoded in seconds since 00:00:00 UTC, 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.

NAME

ucontext – user context

SYNOPSIS

#include <ucontext.h>

DESCRIPTION

The **ucontext** structure defines the context of a thread of control within an executing process.

This structure includes at least the following members:

```
ucontext_t  *uc_link
sigset_t    uc_sigmask
stack_t     uc_stack
mcontext_t  uc_mcontext
```

uc_link is a pointer to the context that is to be resumed when this context returns. If **uc_link** is equal to 0, then this context is the main context, and the process exits when this context returns. The **uc_link** field is only meaningful for contexts created using **makecontext**.

uc_sigmask defines the set of signals that are blocked when this context is active [see **sigprocmask**(2)].

uc_stack defines the stack used by this context [see **sigaltstack**(2)].

uc_mcontext contains the saved set of machine registers and any implementation specific context data. Portable applications should not modify or access **uc_mcontext**.

SEE ALSO

getcontext(2), **makecontext**(3C), **sigaction**(2), **sigaltstack**(2), **sigprocmask**(2)

NAME

values – machine-dependent values

SYNOPSIS

#include <values.h>

DESCRIPTION

This file contains a set of manifest constants, conditionally defined for particular processor architectures.

The model assumed for integers is binary representation (one's or two's complement), where the sign is represented by the value of the high-order bit.

BITS(*type*)	The number of bits in a specified type (for example, **int**).
HIBITS	The value of a short integer with only the high-order bit set.
HIBITL	The value of a long integer with only the high-order bit set.
HIBITI	The value of a regular integer with only the high-order bit set.
MAXSHORT	The maximum value of a signed short integer.
MAXLONG	The maximum value of a signed long integer.
MAXINT	The maximum value of a signed regular integer.
MAXFLOAT, LN_MAXFLOAT	The maximum value of a single-precision floating-point number, and its natural logarithm.
MAXDOUBLE, LN_MAXDOUBLE	The maximum value of a double-precision floating-point number, and its natural logarithm.
MAXLONGDOUBLE, LN_MAXLONGDOUBLE	The maximum value of a quad-precision floating-point number, and its natural logarithm.
MINFLOAT, LN_MINFLOAT	The minimum positive value of a single-precision floating-point number, and its natural logarithm.
MINDOUBLE, LN_MINDOUBLE	The minimum positive value of a double-precision floating-point number, and its natural logarithm.
MINLONGDOUBLE, LN_MINLONGDOUBLE	The minimum value of a quad-precision floating-point number, and its natural logarithm.
FSIGNIF	The number of significant bits in the mantissa of a single-precision floating-point number.
DSIGNIF	The number of significant bits in the mantissa of a double-precision floating-point number.
LDSIGNIF	The number of significant bits in the mantissa of a quad-precision floating-point number.

SEE ALSO

intro(3), **math**(5)

NAME

varargs – handle variable argument list

SYNOPSIS

#include <varargs.h>

va_alist

va_dcl

va_list *pvar*;

void va_start(va_list *pvar*);

type **va_arg(va_list** *pvar*, *type*);

void va_end(va_list *pvar*);

DESCRIPTION

This set of macros allows portable procedures that accept variable argument lists to be written. Routines that have variable argument lists [such as **printf**(3S)] but do not use **varargs** are inherently non-portable, as different machines use different argument-passing conventions.

va_alist is used as the parameter list in a function header.

va_dcl is a declaration for **va_alist**. No semicolon should follow **va_dcl**.

va_list is a type defined for the variable used to traverse the list.

va_start is called to initialize *pvar* to the beginning of the list.

va_arg will return the next argument in the list pointed to by *pvar*. *type* is the type the argument is expected to be. Different types can be mixed, but it is up to the routine to know what type of argument is expected, as it cannot be determined at runtime.

va_end is used to clean up.

Multiple traversals, each bracketed by **va_start** and **va_end**, are possible.

EXAMPLE

This example is a possible implementation of **execl** [see **exec**(2)].

```
#include <unistd.h>
#include <varargs.h>
#define MAXARGS 100

/*    execl is called by
            execl(file, arg1, arg2, . . ., (char *)0);
*/
execl(va_alist)
va_dcl
{
      va_list ap;
      char *file;
      char *args[MAXARGS];          /* assumed big enough*/
      int argno = 0;

      va_start(ap);
```

```
                file = va_arg(ap, char *);
                 while ((args[argno++] = va_arg(ap, char *)) != 0)
                      ;
                 va_end(ap);
                 return execv(file, args);
           }
```

SEE ALSO

exec(2), **printf**(3S), **stdarg**(5), **vprintf**(3S)

NOTES

It is up to the calling routine to specify in some manner how many arguments there are, since it is not always possible to determine the number of arguments from the stack frame. For example, **execl** is passed a zero pointer to signal the end of the list. **printf** can tell how many arguments are there by the format.

It is non-portable to specify a second argument of **char**, **short**, or **float** to **va_arg**, since arguments seen by the called function are not **char**, **short**, or **float**. C converts **char** and **short** arguments to **int** and converts **float** arguments to **double** before passing them to a function.

stdarg is the preferred interface.

NAME

wstat – wait status

SYNOPSIS

#include <sys/wait.h>

DESCRIPTION

When a process waits for status from its children via either the **wait** or **waitpid** function, the status returned may be evaluated with the following macros, defined in **sys/wait.h**. These macros evaluate to integral expressions. The *stat* argument to these macros is the integer value returned from **wait** or **waitpid**.

WIFEXITED(*stat*)	Evaluates to a non-zero value if status was returned for a child process that terminated normally.
WEXITSTATUS(*stat*)	If the value of **WIFEXITED**(*stat*) is non-zero, this macro evaluates to the exit code that the child process passed to **_exit** or **exit**, or the value that the child process returned from **main**.
WIFSIGNALED(*stat*)	Evaluates to a non-zero value if status was returned for a child process that terminated due to the receipt of a signal.
WTERMSIG(*stat*)	If the value of **WIFSIGNALED**(*stat*) is non-zero, this macro evaluates to the number of the signal that caused the termination of the child process.
WIFSTOPPED(*stat*)	Evaluates to a non-zero value if status was returned for a child process that is currently stopped.
WSTOPSIG(*stat*)	If the value of **WIFSTOPPED**(*stat*) is non-zero, this macro evaluates to the number of the signal that caused the child process to stop.
WIFCONTINUED(*stat*)	Evaluates to a non-zero value if status was returned for a child process that has continued.
WCOREDUMP(*stat*)	If the value of **WIFSIGNALED**(*stat*) is non-zero, this macro evaluates to a non-zero value if a core image of the terminated child was created.

SEE ALSO

exit(2), **wait**(2), **waitpid**(2)

NAME

`intro` – introduction to special files

DESCRIPTION

This section describes various special files that refer to specific hardware peripherals, and UNIX system device drivers. STREAMS [see **`intro`**(2)] software drivers, modules and the STREAMS-generic set of **`ioctl`**(2) system calls are also described.

For hardware related files, the names of the entries are generally derived from names for the hardware, as opposed to the names of the special files themselves. Characteristics of both the hardware device and the corresponding UNIX system device driver are discussed where applicable.

Disk device file names are in the following format:

`/dev/{r}dsk/{1.0}s#` For integral disks

`/dev/{r}dsk/c#t#d#s#` For SCSI disks

where **`r`** indicates a raw interface to the disk, the **`c#`** indicates the SCSI host adapter number, **`t#`** indicates the SCSI target *ID* of the device, **`d#`** indicates the device attached to the controller and **`s#`** indicates the section number of the partitioned device.

adsc (7)

NAME

adsc – Adaptec 1542A SCSI host adapter subsystem

DESCRIPTION

The Adaptec 1542A host adapter subsystem enables SCSI target drivers (such as **sd01**, **st01**, and so on) to communicate on the SCSI bus with target controllers and logical units. This driver implements the Portable Device Interface (PDI) for such PDI target drivers.

It is also possible to access the SCSI-bus subsystem directly by using the driver's pass-through interface. This allows you to issue **sb** control blocks directly to the target controller. To find the appropriate pass-through device to use, while any device is being accessed through the target driver (for example,**sd01**), use the **B_GETDEV ioctl** to get the major and minor numbers of the pass-through node. This node may be created and opened for pass-through use (**SDI_SEND ioctl**).

ioctl Calls

The following **ioctl**(2) commands are supported by this driver:

SDI_SEND
: Sends a pass-through command (SCSI control block) to a target controller, bypassing the associated target driver.

SDI_BRESET
: Resets the SCSI bus.

B_REDT
: Reads the extended Equipped Device Table (EDT) data structure that is stored in the **adsc** driver's internal data structure.

B_GETTYPE
: Returns the bus name (for example, SCSI) and device driver name of a specific device.

Files

/usr/include/sys/ad1542.h
/usr/include/sys/scsi.h
/usr/include/sys/sdi.h
/usr/include/sys/sdi_edt.h
/etc/conf/pack.d/adsc/space.c

NOTICES

Adaptec SCSI controllers cannot be used reliably with Emulex SCSI/ESDI bridge controllers (also known as AT&T DCM/4E). In the future, a hardware and/or software fix may be implemented.

REFERENCES

dpt(7), **ioctl**(2), **mcis**(7), **sc01**(7), **sd01**(7), **st01**(7), **sw01**(7), **wd7000**(7)

NAME

alp – algorithm pool management module

DESCRIPTION

The STREAMS module **alp** maintains a pool of algorithms (in the form of STREAMS-compatible subroutines) that may be used for processing STREAMS data messages. Interfaces are defined allowing modules to request and initiate processing by any of the algorithms maintained in the pool. It is expected to help centralize and standardize the interfaces to algorithms that now represent a proliferation of similar-but-different STREAMS modules. Its major use is envisioned as a central registry of available code set conversion algorithms or other types of common data-manipulating routines.

An *algorithm pool* is a registry (or *pool*) of available functions; in this case, routines for performing transformations on STREAMS data messages. Registered functions may keep information on attached users, which means that algorithms need not be stateless, but may maintain extensive state information related to each connection. An algorithm from the pool is called by another in-kernel module with arguments that are a STREAMS data message and a unique identifier. If a message is passed back to the caller, it is the algorithm's output, otherwise the algorithm may store partially convertible input until enough input is received to give back output on a subsequent call.

This pool is one means for providing a consistent and flexible interface for *code set conversion* within STREAMS modules, especially **kbd**, but it may also be used to provide other services that are commonly duplicated by several modules.

The **alp** module contains some subroutines dealing with its (minor) role as a module, a data definition for an algorithm list, connection and disconnection routines, and a search routine for finding registered items. The module interface incorporated into **alp** serves the purpose of providing an **ioctl** interface, so that users can find out what algorithms are registered [see **alpq**(1)].

The programmer of a function for use with **alp** provides a simple module with a simple specified interface. The module must have an initialization routine (*xxx***init**) which is called at system startup time to register itself with **alp**, an open routine, and an interface routine (which actually implements the algorithm).

The registry method of dynamically building the list of available functions obviates the need for recompiling modules or otherwise updating a list or reconfiguring other parts of the system to accommodate additions or deletions. To install a new function module, one merely links it with the kernel in whatever manner is standard for that system; there is no need for updating or re-configuring any other parts of the kernel (including **alp** itself). The remainder of this discussion concerns the in-kernel operation and use of the module.

Calling Sequence

An algorithm is called from the pool by first requesting a connection via the **alp** connection interface. The **alp** module returns the function address of an interface routine, and fills in a unique identifier (**id**) for the connection. The returned function address is NULL on failure (and **id** is undefined). This is a sample of making a connection to a function managed by **alp**:

```
unsigned char *name;     /* algorithm name */
caddr_t id;              /* unique id */
mblk_t *(*func)();       /* func returns ptr to mblk_t */
mblk_t *(*alp_con())(); /* returns pointer to mblk_t */
...
if (func = alp_con(name, (caddr_t) &id))
     regular processing;
else
     error processing;
```

Once the connection has been made, the interface routine can be called directly by the connecting module to process messages:

```
mblk_t *inp, *outp;
mblk_t *(*func)();
...
outp = (*func)(mp, id);
mp = NULL;    /* mp cannot be re-used! */
if (outp)
     regular processing;
```

If the interface routine processed the entire message, then **outp** is a valid pointer to the algorithm's output message. If, however, the routine needs more information, or is buffering something, **outp** will be a null pointer. In either case, the original message (**mp**) may not be subsequently accessed by the caller. The interface routine takes charge of the message **mp**, and may free it or otherwise dispose of it (it may even return the same message). The caller may pass a null message pointer to an interface routine to cause a flush of any data being held by the routine; this is useful for end-of-file conditions to insure that all data have been passed through. (Interface routines must thus recognize a null message pointer and deal with it.)

Synchronization between input and output messages is not guaranteed for all items in the pool. If one message of input does not produce one message of output, this fact should be documented for that particular module. Many multibyte code set conversion algorithms, to cite one instance, buffer partial sequences, so that if a multibyte character happens to be spread across more than one message, it may take two or more output messages to complete translation; in this case, it is only possible to synchronize when input message boundaries coincide with character boundaries.

Building an Algorithm for the Pool

As mentioned, the modules managed by **alp** are implemented as simple modules—not STREAMS modules—each with an initialization routine, an open routine, and a user-interface routine. The initialization routine is called when the system is booted and prior to nearly everything else that happens at boot-time. The routine takes no arguments and its sole purpose is to register the algorithm with the **alp** module, so that it may subsequently accessed. Any other required initialization may also be performed at that time. A generic initialization routine for a module called **GEN**, with prefix **gen** is as follows:

```
geninit()
{
      mblk_t *genfunc(); /* interface routine */
      int rval;          /* return value from registrar */

      rval = alp_register(genfunc, "name", "explanation");
      if (rval) cmn_err(CE_WARN, "warning message");
}
```

The registration routine, **alp_register** takes three arguments and returns zero if successful. The arguments are (1) a pointer to the algorithm's entry point (in this case, the function **genfunc**), (2) a pointer to its name, and (3) a pointer to a character string containing a brief explanation. The name should be limited to under 16 bytes, and the explanation to under 60 bytes, as shown in the following example. Neither the name nor the explanation need include a newline.

```
i = alp_register(sjisfunc, "stou",
      "Shift-JIS to UJIS converter");
```

It is possible for a single module to contain several different, related algorithms, which can each be registered separately by a single **init** routine.

A module's open routine is called by **alp_con** when a connection is first requested by a user (that is, a module that wishes to use it). The open routine takes two arguments. The first argument is an integer; if it is non-zero, the request is an open request, and the second argument is unused. The function should allocate a unique identifier and return it as a generic address pointer. If the first argument is zero, the request is a close request, and the second argument is the unique identifier that was returned by a previous open request, indicating which of (potentially several) connections is to be closed. The routine does any necessary clean-up and closes the connection; thereafter, any normal interface requests on that identifier will fail. This use of unique identifiers allows these modules to keep state information relating to each open connection; no format is imposed upon the unique identifier, so it may contain any arbitrary type of information, equivalent in size to a core address; **alp** and most callers will treat it as being of type **caddr_t**, in a manner similar to the private data held by each instantiation of a STREAMS module.

A skeleton for the **gen** module's open routine is:

```
genopen(arg, id)
      int arg;
      caddr_t id;
{
      if ( arg ) {
            open processing;
            return( unique-id );
      }
      close processing for id;
      return(0);
}
```

Once a connection has been made, users may proceed as in the example in the previous section. When the connection is to be closed (for example, the connecting module is being popped), a call is made to **alp_discon**, passing the unique id and the name:

```
caddr_t id;
char *name;
mblk_t *alp_discon(), *mp;
...
mp = alp_discon(name, id);
if (mp)
        process "left-over" data;
```

If the disconnect request returns a valid message pointer (**mp**) then there was unprocessed or partially processed data left in an internal buffer, and it should be dealt with by the caller (for example, by flushing it or sending it to the neighboring module).

The ioctl and Query Interfaces

A kernel-level query interface is provided in addition to the query interface supported by the **alpq** command. The routine **alp_query** takes a single argument, a pointer to a *name*. If the name matches a registered function, **alp_query** returns a pointer to the function's *explanation* string, otherwise it returns a null pointer. A calling example is:

```
unsigned char *alp_query(), *name, *expl;
...
if (expl = alp_query(name))
        regular processing;
else
        error processing;
```

The `ioctl` interface provides calls for querying registered functions (for which the *explanation* discussed above is necessary); this is supported by the **alpq** command, which may be used whenever user-level programs need the associated information.

Uses

The **alp** module can be used to replace various kernel-resident code set conversion functions in international or multi-language environments. The KBD subsystem (which supplies code set conversion and keyboard mapping) supports the use of **alp** algorithms as processing elements.

Since state information may be maintained, functions may also implement processing on larger or more structured data elements, such as transaction records and network packets. Currently, STREAMS CPU priority is assumed by **alp** or should be set individually by interface and open routines.

EXAMPLES

```
/*
 * This is a SAMPLE module that registers with ALP and
 * performs a one-message delay.
 */
#include <sys/types.h>
#include <sys/stream.h>
```

```
#include <sys/stropts.h>
#include <sys/kmem.h>
#include <sys/alp.h>

static mblk_t *dely();
caddr_t delyopen();

/*
 * Our state structure.  Keeps its own address and a pointer.
 */
struct dstruct {
     caddr_t d_unique;
     mblk_t *d_mp;
};

/*
 * The name is "Dely".  It has an open routine "delyopen"
 * and an interface "dely".
 */
static struct algo delyalgo =
{
     0, (queue_t *) 0, (queue_t *) 0, dely, delyopen,
     (unsigned char *) "Dely",
     (unsigned char *) "One Message Delay Buffer",
     (struct algo *) 0
};

/*
 * This is the sysinit routine, called when the system is
 * being brought up.  It registers "Dely" with ALP.
 */
delyinit()
{
     if (alp_register(&delyalgo))     /* then register with ALP */
          printf("DELY: register failed\n");
}
/*
 * This is the interface routine itself.
 * Holds onto "mp" and returns whatever it had before.
 */
static mblk_t *
dely(mp, id)
     mblk_t *mp;
     caddr_t id;
{
     register mblk_t *rp;
     register struct dstruct *d;

     d = (struct dstruct *) id; /* clarify the situation */
```

```
          rp = d->d_mp;
          d->d_mp = mp;
          return(rp);           /* return the previous message */
     }

     /*
      * The open (and close) routine.  Use kmem_alloc() to get
      * a private structure for saving state info.
      */
     caddr_t
     delyopen(arg, id)
          int arg;    /* 1 = open, 0 = close */
          caddr_t id; /* ignored on open; unique id on close */
     {
          register struct dstruct *d;
          register mblk_t *rp;

          if (! arg) {    /* close processing */
               d = (struct dstruct *) id;
               d->d_unique = (caddr_t) -1;
               rp = d->d_mp;
               kmem_free(d, sizeof(struct dstruct));
               return((caddr_t) rp);
          }
          /* otherwise, open processing */
          d = (struct dstruct *) kmem_zalloc(sizeof(struct dstruct),
                    KM_NOSLEEP);
          d->d_unique = (caddr_t) &d;
          return((caddr_t) d);
     }
```

SEE ALSO

alpq(1), kbd(7)

NAME

ARP – Address Resolution Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <net/if_arp.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_DGRAM, 0);

d = open ("/dev/arp", O_RDWR);
```

DESCRIPTION

ARP is a protocol used to map dynamically between Internet Protocol (IP) and 10Mb/s Ethernet addresses. It is used by all the 10Mb/s Ethernet datalink providers (interface drivers). It is not specific to the Internet Protocol or to the 10Mb/s Ethernet, but this implementation currently supports only that combination. The STREAMS device **/dev/arp** is not a Transport Level Interface (TLI) transport provider and may not be used with the TLI interface.

ARP caches IP-to-Ethernet address mappings. When an interface requests a mapping for an address not in the cache, ARP queues the message that requires the mapping and broadcasts a message on the associated network requesting the address mapping. If a response is provided, the new mapping is cached and any pending message is transmitted. ARP will queue at most one packet while waiting for a mapping request to be responded to; only the most recently transmitted packet is kept.

To facilitate communications with systems which do not use ARP, **ioctl** requests are provided to enter and delete entries in the IP-to-Ethernet tables.

USAGE

```
#include <sys/sockio.h>
#include <sys/socket.h>
#include <net/if.h>
#include <net/if_arp.h>

struct arpreq arpreq;

ioctl(s, SIOCSARP, (caddr_t)&arpreq);
ioctl(s, SIOCGARP, (caddr_t)&arpreq);
ioctl(s, SIOCDARP, (caddr_t)&arpreq);
```

Each **ioctl** request takes the same structure as an argument. **SIOCSARP** sets an ARP entry, **SIOCGARP** gets an ARP entry, and **SIOCDARP** deletes an ARP entry. These **ioctl** requests may be applied to any Internet family socket descriptor *s*, or to a descriptor for the ARP device, but only by the privileged user. The **arpreq** structure contains:

```
/*
* ARP ioctl request
*/
struct arpreq {
     struct sockaddr arp_pa;     /* protocol address */
     struct sockaddr arp_ha;     /* hardware address */
     int             arp_flags;  /* flags */
};
/*  arp_flags field values */
#define ATF_INUSE        0x01 /* entry in use */
#define ATF_COM          0x2  /* completed entry (arp_ha valid) */
#define ATF_PERM         0x4  /* permanent entry */
#define ATF_PUBL         0x8  /* publish (respond for other host) */
#define ATF_USETRAILERS  0x10 /* send trailer packets to host */
```

The address family for the **arp_pa sockaddr** must be **AF_INET**; for the **arp_ha sockaddr** it must be **AF_UNSPEC**. The only flag bits that may be written are **ATF_PERM**, **ATF_PUBL** and **ATF_USETRAILERS**. **ATF_PERM** makes the entry permanent if the **ioctl** request succeeds. The peculiar nature of the ARP tables may cause the **ioctl** request to fail if too many permanent IP addresses hash to the same slot. **ATF_PUBL** specifies that the ARP code should respond to ARP requests for the indicated host coming from other machines. This allows a host to act as an ARP server, which may be useful in convincing an ARP-only machine to talk to a non-ARP machine.

ARP is also used to negotiate the use of trailer IP encapsulations; trailers are an alternate encapsulation used to allow efficient packet alignment for large packets despite variable-sized headers. Hosts that wish to receive trailer encapsulations so indicate by sending gratuitous ARP translation replies along with replies to IP requests; they are also sent in reply to IP translation replies. The negotiation is thus fully symmetrical, in that either or both hosts may request trailers. The **ATF_USETRAILERS** flag is used to record the receipt of such a reply, and enables the transmission of trailer packets to that host.

ARP watches passively for hosts impersonating the local host (that is, a host which responds to an ARP mapping request for the local host's address).

SEE ALSO

arp(1M), **if**(7), **ifconfig**(1M), **inet**(7)

Plummer, Dave, "*An Ethernet Address Resolution Protocol -or- Converting Network Protocol Addresses to 48.bit Ethernet Addresses for Transmission on Ethernet Hardware*," RFC 826, Network Information Center, SRI International, Menlo Park, Calif., November 1982

Leffler, Sam, and Michael Karels, "*Trailer Encapsulations*," RFC 893, Network Information Center, SRI International, Menlo Park, Calif., April 1984

NAME

asyc – asynchronous serial port

DESCRIPTION

The **asyc** driver supports both the system board serial port and an additional serial adapter simultaneously. Up to two serial ports are supported. If an adapter for a port is not installed, an attempt to open it will fail. Depending on your system processor type and UART, the port can be programmed for speed (50–38400 baud), character length, and parity. Output speed is always the same as input speed. The port behaves as described in **termio** (7).

The asynchronous port is a character-at-a-time device for both input and output. This characteristic both limits the bandwidth that can be achieved over a line, and increases the interrupt loading on the central processor. File transfer programs such as **uucp**(1C) may be able to function well at speeds greater than 9600 baud, depending on your system processor type and UART.

The baud rates of the serial adapter programmable baud-rate generator do not correspond exactly with system baud rates. In particular, setting B0 will cause a disconnect, setting EXTA will set 19200 baud, and setting EXTB will set 38400 baud. It is not possible to directly set 2000, 3600, or 7200 baud. The asynchronous ports driver supports line signal (hardware) flow control when the device node **/dev/tty0?h** is used. The **/dev/tty0?s** ports are software flow control nodes as are the **/dev/tty0?** nodes.

FILES

/dev/tty* /dev/term/*

SEE ALSO

signal(2), **termio**(7)

clone(7)

NAME

clone – open any major/minor device pair on a STREAMS driver

DESCRIPTION

clone is a STREAMS software driver that finds and opens an unused major/minor device on another STREAMS driver. The major device number passed to **clone** during open corresponds to the **clone** driver and the minor device number corresponds to the target driver. Each open results in a separate stream to a previously unused major/minor device.

The **clone** driver consists solely of an open function. This open function performs all of the necessary work so that subsequent system calls [including **close**(2)] require no further involvement of **clone**.

clone will generate an **ENXIO** error, without opening the device, if the major/minor device number provided does not correspond to a valid major/minor device, or if the driver indicated is not a STREAMS driver.

NOTICES

Multiple opens of the same major/minor device cannot be done through the **clone** interface. Executing **stat**(2) on the file system node for a cloned device yields a different result from executing **fstat** using a file descriptor obtained from opening the node.

REFERENCES

log(7)

NAME

connld – line discipline for unique stream connections

DESCRIPTION

connld is a STREAMS-based module that provides unique connections between server and client processes. It can only be pushed [see **streamio**(7)] onto one end of a STREAMS-based pipe that may subsequently be attached to a name in the file system name space. After the pipe end is attached, a new pipe is created internally when an originating process attempts to **open**(2) or **creat**(2) the file system name. A file descriptor for one end of the new pipe is packaged into a message identical to that for the **ioctl I_SENDFD** [see **streamio**(7)] and is transmitted along the stream to the server process on the other end. The originating process is blocked until the server responds.

The server responds to the **I_SENDFD** request by accepting the file descriptor through the **I_RECVFD ioctl** message. When this happens, the file descriptor associated with the other end of the new pipe is transmitted to the originating process as the file descriptor returned from **open**(2) or **creat**(2).

If the server does not respond to the **I_SENDFD** request, the stream that the **connld** module is pushed on becomes uni-directional because the server will not be able to retrieve any data off the stream until the **I_RECVFD** request is issued. If the server process exits before issuing the **I_RECVFD** request, the **open**(2) or the **creat**(2) system calls will fail and return -1 to the originating process.

When the **connld** module is pushed onto a pipe, messages going back and forth through the pipe are ignored by **connld**.

On success, an open of **connld** returns 0. On failure, **errno** is set to the following values:

EINVAL	A stream onto which **connld** is being pushed is not a pipe or the pipe does not have a write queue pointer pointing to a stream head read queue.
EINVAL	The other end of the pipe onto which **connld** is being pushed is linked under a multiplexor.
EPIPE	**connld** is being pushed onto a pipe end whose other end is no longer there.
ENOMEM	An internal pipe could not be created.
ENXIO	An **M_HANGUP** message is at the stream head of the pipe onto which **connld** is being pushed.
EAGAIN	Internal data structures could not be allocated.
ENFILE	A file table entry could not be allocated.

SEE ALSO

streamio(7)

console (7)

NAME

console – STREAMS-based console interface

DESCRIPTION

The file **/dev/console** is the system console and refers to an asynchronous serial data line originating from the system board.

The file **/dev/contty** refers to a second asynchronous serial data line originating from the system board.

Both **/dev/console** and **/dev/contty** access the STREAMS-based console driver, which when used in conjunction with the STREAMS line discipline module **ldterm**, supports the **termio**(7) and **termios**(2) processing.

FILES

/dev/console
/dev/contty

SEE ALSO

crash(1M), **ldterm**(7), **termio**(7), **termios**(2)

NAME

`cram` – CMOS RAM interface

DESCRIPTION

The cram driver provides an interface to the 64 bytes of battery backed-up RAM. This memory contains information such as diagnostics and configuration information.

ioctl Calls

`CMOSREAD`

This call is used to read the contents of one of the CMOS RAM locations. The argument to the `ioctl` is the address of a buffer of two unsigned characters, the first of which is the address to be read. The `ioctl` will fill in the second byte with the data. An address less than 0 or greater than 63 will result in an error, with **`errno`** set to **`ENXIO`**.

`CMOSWRITE`

This call is used to write a value into one of the CMOS RAM locations. The argument to the `ioctl` is the address of a buffer of two unsigned characters, the first of which is the address and the second of which is the value to write at that address. An address less than 0 or greater than 63 will result in an error, with **`errno`** set to **`ENXIO`**. Note that only the super-user may open the CMOS RAM device for writing, and that the **`CMOSWRITE`** `ioctl` will fail for any other than the super-user.

Files

`/dev/cram`

NAME

DCD – Direct-Coupled Disk host adapter Subsystem

DESCRIPTION

The DCD subsystem consists of at least one DCD Host Controller card which has at least one logical unit attached to it.

The DCD subsystem adds support for non-SCSI devices under SDI for use with related target drivers. This subsystem is accessed indirectly by opening an appropriate target driver to access a target device that is on a DCD controller.

It is also possible to access this subsystem via the pass-through driver. To find the appropriate device to use, while the device is being accessed through the target driver, use the **B_GETDEV ioctl** to get the major and minor numbers of the pass-though node. This node may either be created and/or open for use.

There are four groups of **ioctl**(2) commands supported by DCD . The first group contains the **ioctl** commands used by the DCD driver itself.

SDI_SEND	Used to send a pass through command to a target controller bypassing the associated target driver.
SDI_BRESET	Used to reset the bus.
HA_VER	For a 80386-based computer, used to determine the Driver Interface Version supported by the driver. It returns the structure **ver_no** defined in **sdi.h**.

The second group is used by the driver and all target drivers that use the SCSI Driver Interface protocol to communicate with their associated target controllers.

B_GETTYPE	Return the bus name (DCD) and device driver name of a specific device.

The third group should be used by all the target drivers that use the SCSI Driver Interface protocol to communicate with their associated target controllers. This **ioctl** is not supported by the DCD driver.

B_GETDEV	Return the pass through major and minor numbers to the calling utility to allow creation of a pass through the special device file.

The fourth group should be used to get and set device geometry. These return or accept the structure **dsk_geom** defined in **sdi.h**.

HA_GETPARMS	Return Host adapters idea of the device geometry. This is what the system uses during the boot sequence. This is used by certain target drivers.
HA_SETPARMS	Set the Host adapters idea of the device geometry. Note that some Host adapters do not support this **ioctl** and will result in an error.
HA_GETPPARMS	Return the actual device geometry. This is the actual as opposed to virtual device geometry.

The DCD driver has a halt routine that is executed during shutdown, which allows the controller enough time to flush the disk cache (if present) to disk. Although two seconds are allowed for this flush, the actual timeout value is an external variable in the DCD **space.c** file. You can modify this variable if a specific controller needs more time to clear the cache.

Files

`/usr/include/sys/sdi.h`
`/usr/include/sys/sdi_edt.h`

REFERENCES

`ioctl`(2)

NAME

display – system console display

SYNOPSIS

```
#include <sys/console.h>
#include <sys/kd.h>
#include <sys/vt.h>
```

DESCRIPTION

The system console (and user's terminal) is composed of two separate pieces: the keyboard [see **keyboard** (7)] and the display. Because of their complexity, and because there are three possible display interfaces (monochrome, color graphics, and enhanced graphics adapters), they are discussed in separate manual entries.

The display normally consists of 25 lines of 80 columns each; 40-column lines are also supported by the color/graphics adapter, and 43 lines of 80-columns each are supported by the enhanced graphics adapter. When characters are written to the console or one of its virtual screens (**/dev/console** or **/dev/vt***xx*), the output depends on the specific characters. All characters written to **/dev/console** are first processed by the terminal interface [see **termio** (7)]. For example, mapping new-line characters to carriage return plus new-line, and expanding tabs to spaces, will be done before the following processing:

x	Where **x** is not one of the following, displays **x**.
BEL	Generates a bell (audible tone, no modulation).
CR	Places the cursor at column 1 of the current line.
LF, VT	Places the cursor at the same column of the next line (scrolls if the the current line is line 25).
FF	Clears the screen and places the cursor at line 1, column 1.
BS	If the cursor is not at column 1, it is moved to the left one position on the same line. If the cursor is at column 1 but not line 1, it is moved to column 79 of the previous line. Finally, if the cursor is at column 1, line 1, it is not moved.

The display can be controlled by using specific sequences of characters that are ANSI X3.64 *escape sequences* preceded by the ASCII character ESC. The escape sequences, which work on either the monochrome, color graphics, or enhanced graphics adapter, are the following:

ESC c Clears the screen and places the cursor at line 1, column 1.

ESC Q *n 'string'*
Defines the function key *n* with *string*. The string delimiter ′ may be any character not in *string*. Function keys are numbered 0 through 11 (F1 = 0, F2 = 1, and so on.)

ESC [*n* **@** Insert character—inserts *n* blanks at the current cursor position.

ESC [*n* **`** Horizontal Position Absolute—moves active position to column given by *n*.

ESC[*n* **A**	Cursor up—moves the cursor up *n* lines (default: *n*=1).
ESC[*n* **a**	Horizontal Position Relative—moves active position *n* characters to the right (default: *n*=1).
ESC[*n* **B**	Cursor down—moves the cursor down *n* lines (default: *n*=1).
ESC[*n* **C**	Cursor right—moves the cursor right *n* columns (default: *n*=1).
ESC[*n* **c**	Set Cursor Type—where n is 0 (underline cursor), 1(blockcursor), or 2(no cursor). 0 is the default value for n.
ESC[*n* **D**	Cursor left—moves the cursor left *n* columns (default: *n*=1).
ESC[*n* **d**	Vertical Position Absolute—moves active position to line given by *n*.
ESC[*n* **E**	Cursor next line—moves the cursor to column 1 of the next line, then down *n*–1 lines (default: *n*=1).
ESC[*n* **e**	Vertical Position Relative—moves the active position down *n* lines (default: *n*=1).
ESC[*n* **F**	Cursor previous line—moves the cursor to column 1 of the current line, then up *n* lines (default: *n*=1).
ESC[*n* **G**	Cursor horizontal position—moves the cursor to column *n* of the current line (default: *n*=1).
ESC[*n* ; *m* **H**	Position cursor—moves the cursor to column *m* of line *n* (default: *n*=1, *m*=1).
ESC[*n* ; *m* **f**	Position cursor—moves the cursor to column *m* of line *n* (default: *n*=1, *m*=1).
ESC[*n* **J**	Erase window—erases from the current cursor position to the end of the window if *n*=0, from the beginning of the window to the current cursor position if *n*=1, and the entire window if *n*=2 (default: *n*=0).
ESC[*n* **K**	Erase line—erases from the current cursor position to the end of the line if *n*=0, from the beginning of the line to the current cursor position if *n*=1, and the entire line if *n*=2 (default: *n*=0).
ESC[*n* **L**	Insert line—inserts *n* lines at the current cursor position (default: *n*=1).
ESC[*n* **M**	Delete line—deletes *n* lines starting at the current cursor position (default: *n*=1).
ESC[*n* **P**	Delete character—deletes *n* characters from a line starting at the current cursor position (default: *n*=1).
ESC[*n* **S**	Scroll up—scrolls the characters in the current window up *n* lines. The bottom *n* lines are cleared to blanks (default: *n*=1).
ESC[*n* **T**	Scroll down—scrolls the characters in the current window down *n* lines. The top *n* lines are cleared to blanks (default: *n*=1).
ESC[*n* **X**	Erase character—erases *n* character positions starting at the current cursor position (default: *n*=1).

`ESC[` *n* `Z` Cursor Backward Tabulation—moves active position back *n* tab stops.

`ESC[ 2 h` Locks the keyboard and ignores keyboard input until unlocked. Characters are not saved.

`ESC[ 2 i` Sends the screen to the host. The current screen display is sent to the application.

`ESC[ 2 l` Unlocks the keyboard. Re-enables keyboard input.

`ESC[` *Ps* ; *Ps*; `m`

Character attributes—each *Ps* is a value in the table below; multiple characters are separated by semicolons. These parameters apply to successive characters being displayed, in an additive way (for example, both bold and underscoring can be selected). Only the parameters through 7 apply to the monochrome adapter; all parameters apply to the color or graphics adapter and the enhanced graphics adapter. (Default: *Ps*=0).

Ps	Meaning		
0	all attributes off (normal display) (white foreground with black background)		
1	bold intensity		
4	underscore on (white foreground with red background on color)		
5	blink on		
6	VGA only: if blink (5) is on, turn blink off and background color to its light equivalent (that is, brown to yellow).		
7	reverse video		
30	black	(gray)	foreground
31	red	(light red)	foreground
32	green	(light green)	foreground
33	brown	(yellow)	foreground
34	blue	(light blue)	foreground
35	magenta	(light magenta)	foreground
36	cyan	(light cyan)	foreground
37	white	(bright white)	foreground
40	black	(gray)	background
41	red	(light red)	background
42	green	(light green)	background
43	brown	(yellow)	background
44	blue	(light blue)	background
45	magenta	(light magenta)	background
46	cyan	(light cyan)	background
47	white	(bright white)	background

Note that for character attributes 30–37, the color selected for foreground will depend on whether the *bold intensity* attribute (1) is currently on. If not, the first color listed will result; otherwise the second color listed will result.

Similarly, for character attributes 40–47, the color selected for background will depend on whether the *blink* attribute (5) is currently on and bright background (6) has been turned on. If *blink* is not turned on or bright background has not been selected, the first listed color will result. Otherwise, the second color listed will result.

`ESC [ 8 m`	sets blank (non-display)
`ESC [ 10 m`	selects the primary font
`ESC [ 11 m`	selects the first alternate font; lets ASCII characters less than 32 be displayed as ROM characters
`ESC [ 12 m`	selects a second alternate font; toggles high bit of extended ASCII code before displaying as ROM characters
`ESC [ 38 m`	enables underline option; white foreground with white underscore (see the NOTES section)
`ESC [ 39 m`	disables underline option (see the NOTES section)

The following non-ANSI X3.64 escape sequences are supplied:

ESC [= *c* **A**	Sets overscan color.
ESC [= *p* ; *d* **B**	Sets bell parameters (where *p* is the pitch in Hz and *d* is the duration in milliseconds)
ESC [= *s* ; *e* **C**	Sets cursor parameters (where *s* is the starting and *e* is the ending scanlines of the cursor).
ESC [= *x* **D**	Enables/disables intensity of background color (where *x* is 0 for enable and 1 for disable).
ESC [= *x* **E**	Sets/clears blink versus bold background (where *x* is 0 for set and 1 for clear).
ESC [= *c* **F**	Sets normal foreground color. See **GIO_ATTR** for the valid values for *c*.
ESC [= *c* **G**	Sets normal background color. See **GIO_ATTR** for the valid values for *c*.
ESC [= *n* **g**	Displays graphic character *n*.
ESC [= *c* **H**	Sets reverse foreground color. See **GIO_ATTR** for the valid values for *c*.
ESC [= *c* **I**	Sets reverse background color. See **GIO_ATTR** for the valid values for *c*.
ESC [= *c* **J**	Sets graphic foreground color. See **GIO_ATTR** for the valid values for *c*.
ESC [= *c* **K**	Sets graphic background color. See **GIO_ATTR** for the valid values for *c*.
ESC [*n* **z**	Makes virtual terminal number *n* active.
ESC 7	Saves cursor position.

`ESC 8`	Restores cursor position to saved value.
`ESC[ 0 k`	Disables the key-click feature (the default).
`ESC[ 1 k`	Enables the key-click feature. A tone is produced for each key press.

ioctl Calls

The following **`ioctl`** calls may be used to change the display used for the video monitor. If the virtual terminal has not been put in process mode (see the VT_SETMODE ioctl), setting the display mode to a non-text mode will turn off VT switching. VT switches will be re-enabled after the display mode has been reset to a text mode.

Note: All the following ioctls are performed on either a file descriptor to the virtual terminals or to the special file **`/dev/video`**. **`ioctls`** to **`/dev/video`** are indicated with an asterisk (*). For the ioctls to **`/dev/video`** to work, the controlling **`tty`** for the process must be the virtual terminal on which the operation is to be performed. If the **`tty`** is not a virtual terminal, the return value will be -1 and **`errno`** will be set to **`EINVAL`**.

SWAPMONO (*)
: This call selects the monochrome adapter as the output device for the system console.

SWAPCGA (*) This call selects the color/graphics adapter as the output device for the system console.

SWAPEGA (*) This call selects the enhanced graphics adapter as the output device for the system console.

SWAPVGA (*) This call selects the video graphics array as the output device for the system console.

The following **`ioctl`** call may be used to obtain more information about the display adapter currently attached to the video monitor:

CONS_CURRENT (*)
: This call returns the display adapter type currently attached to the video monitor. The return value can be one of: **MONO**, **CGA**, or **EGA**.

The following **`ioctl`** calls may be used to switch display modes on the various video adapters:

SW_B40x25 (*)
: This call selects 40x25 (40 columns by 25 rows) black and white text display mode. It is valid only for CGA and EGA devices.

SW_C40x25 (*)
: This call selects 40x25 (40 columns by 25 rows) color text display mode. It is valid only for CGA and EGA devices.

SW_B80x25 (*)
: This call selects 80x25 (80 columns by 25 rows) black and white text display mode. It is valid only for CGA and EGA devices.

`SW_C80x25` (*)
: This call selects 80x25 (80 columns by 25 rows) color text display mode. It is valid only for CGA and EGA devices.

`SW_BG320` (*)
: This call selects 320x200 black and white graphics display mode. It is valid only for CGA and EGA devices.

`SW_CG320` (*)
: This call selects 320x200 color graphics display mode. It is valid only for CGA and EGA devices.

`SW_BG640` (*)
: This call selects 640x200 black and white graphics display mode. It is valid only for CGA and EGA devices.

`SW_CG320_D` (*)
: This call selects EGA support for 320x200 graphics display mode (EGA mode D). It is valid only for EGA devices.

`SW_CG640_E` (*)
: This call selects EGA support for 640x200 graphics display mode (EGA mode E). It is valid only for EGA devices.

`SW_EGAMONOAPA` (*)
: This call selects EGA support for 640x350 graphics display mode (EGA mode F). It is valid only for EGA devices.

`SW_ENH_MONOAPA2` (*)
: This call selects EGA support for 640x350 graphics display mode with extended memory (EGA mode F*). It is valid only for EGA devices.

`SW_CG640x350` (*)
: This call selects EGA support for 640x350 graphics display mode (EGA mode 10). It is valid only for EGA devices.

`SW_ENH_CG640` (*)
: This call selects EGA support for 640x350 graphics display mode with extended memory (EGA mode 16). It is valid only for EGA devices.

`SW_EGAMONO80x25` (*)
: This call selects EGA monochrome text display mode (EGA mode 7), which emulates support provided by the monochrome adapter. It is valid only for EGA devices.

`SW_ENHB40x25` (*)
: This call selects enhanced 40x25 black and white text display mode. It is valid only for EGA devices.

`SW_ENHC40x25` (*)
: This call selects enhanced 40x25 color text display mode. It is valid only for EGA devices.

`SW_ENHB80x25` (*)
: This call selects enhanced 80x25 black and white display mode. It is valid only for EGA devices.

SW_ENHC80x25 (*)
This call selects enhanced 80x25 color text display mode. It is valid only for EGA devices.

SW_ENHB80x43 (*)
This call selects enhanced 80x43 black and white text display mode. It is valid only for EGA devices.

SW_ENHC80x43 (*)
This call selects enhanced 80x43 color text display mode. It is valid only for EGA devices.

SW_MCAMODE (*)
This call reinitializes the monochrome adapter. It is valid only for monochrome adapters.

SW_ATT640 (*)
This call selects 640x400 16 color mode, when an AT&T Super-Vu video controller is attached.

Switching to an invalid display mode for a display device will result in an error.

The following **ioctls** may be used to obtain information about the current display modes:

CONS_GET (*)
This call returns the current display mode setting for whatever display adapter is being used. Possible return values include:

M_B40x25 (0), black and white 40 columns. CGA and EGA only.

M_C40x25 (1), color 40 columns. CGA and EGA only.

M_B80x25 (2), black and white 80 columns. CGA and EGA only.

M_C80x25 (3), color 80 columns. CGA and EGA only.

M_BG320 (4), black and white graphics 320 by 200. CGA and EGA only.

M_CG320 (5), color graphics 320 by 200. CGA and EGA only.

M_BG640 (6), black and white graphics 640 by 200 high-resolution. CGA and EGA only.

M_EGAMONO80x25 (7), EGA-mono 80 by 25. EGA only.

M_CG320_D (13), EGA mode D.

M_CG640_E (14), EGA mode E.

M_EFAMONOAPA (15), EGA mode F.

M_CG640x350 (16), EGA mode 10.

M_ENHMONOAPA2 (17), EGA mode F with extended memory.

M_ENH_CG640 (18), EGA mode 16.

M_ENH_B40x25 (19), EGA enhanced black and white 40 columns.

M_ENH_C40x25 (20), EGA enhanced color 40 columns.

M_ENH_B80x25 (21), EGA enhanced black and white 80 columns.

M_ENH_C80x25 (22), EGA enhanced color 80 columns.

M_ENH_B80x43 (0x70), EGA black and white 80 by 43.

M_ENH_C80x43 (0x71), EGA color 80 by 43.

M_MCA_MODE (0xff), monochrome adapter mode.

MCA_GET (*) This call returns the current display mode setting of the monochrome adapter. See **CONS_GET** for a list of return values. If the monochrome adapter is not installed, the call will fail and **errno** will be set to 22 (**EINVAL**).

CGA_GET (*) This call returns the current display mode setting of the color/graphics adapter. See **CONS_GET** for a list of return values. If the color graphics adapter is not installed, the call will fail and **errno** will be set to 22 (**EINVAL**).

EGA_GET (*) This call returns the current display mode setting of the enhanced graphics adapter. See **CONS_GET** for a list of return values. If the enhanced graphics adapter is not installed, the call will fail and **errno** will be set to 22 (**EINVAL**).

The following **ioctl** calls may be used to map the video adapter's memory into the user's data space.

MAPCONS (*) This call maps the display memory of the adapter currently being used into the user's data space.

MAPMONO (*) This call maps the monochrome adapter's display memory into the user's data space.

MAPCGA (*) This call maps the color/graphics adapter's display memory into the user's data space.

MAPEGA (*) This call maps the enhanced graphics adapter's display memory into the user's data space.

MAPVGA (*) This call maps the video graphics array's display memory into the user's data space.

You can use **ioctl** calls to input a byte from the graphics adapter port or to output a byte to the graphics adapter port. The argument to the **ioctl** uses the *port_io_arg* data structure:

```
struct port_io_arg {
            struct port_io_struc_args[4];
};
```

The previous example shows that the *port_io_arg* structure points to an array of four *port_io_struc* data structures. The *port_io_struc* has the following format:

```
struc port_io_struc {
      char   dir;          /*direction flag (in vs. out)*/
      ushort port;  /*port address*/
      char   data;         /*byte of data*/
};
```

You can specify one, two, three, or four of the *port_io_struc* structures in the array for one **ioctl** call. The value of *dir* can be either IN_ON_PORT (to specify a byte being input from the graphics adapter port) or OUT_ON_PORT (to specify a byte being output to the graphics adapter port). *Port* is an integer specifying the port address of the desired graphics adapter port. *Data* is the byte of data being input or output as specified by the call. If you are not using any of the *port_io_struc* structures, load the *port* with 0, and leave the unused structures at the end of the array. Refer to your hardware manuals for port addresses and functions for the various adapters.

The following **ioctl** calls may be used to input or output bytes on the graphics adapter port:

MCAIO (*) This call inputs or outputs a byte on the monochrome adapter port as specified.

CGAIO (*) This call inputs or outputs a byte on the color/graphics adapter port as specified.

EGAIO (*) This call inputs or outputs a byte on the enhanced graphics adapter port as specified.

VGAIO (*) This call inputs or outputs a byte on the video graphics array port as specified.

To input a byte on any of the graphics adapter ports, load *dir* with IN_ON_PORT and load *port* with the port address of the graphics adapter. The byte input from the graphics adapter port will be returned in *data*.

To output a byte, load *dir* with OUT_ON_PORT, load *port* with the port address of the graphics adapter, and load *data* with the byte you want to output to the graphics adapter port.

The following **ioctls** may be used with either the monochrome, color graphics, or enhanced graphics adapters:

GIO_FONT8x8 (*)
This call gets the current 8x8 font in use.

GIO_FONT8x14 (*)
This call gets the current 8x14 font in use.

GIO_FONT8x16 (*)
This call gets the current 8x16 font in use.

KDDISPTYPE (*)
This call returns display information to the user. The argument expected is the buffer address of a structure of type *kd_disparam* into which display information is returned to the user. The *kd_disparam* structure is defined as follows:

```
struct kd_disparam {
      long type;     /*display type*/
      char *addr;    /*display memory address*/
      ushort ioaddr[MKDIOADDR];    /*valid I/O addresses*/
}
```

Possible values for the type field include:

KD_MONO (0x01), for the IBM monochrome display adapter.

KD_HERCULES (0x02), for the Hercules monochrome graphics adapter.

KD_CGA (0x03), for the IBM color graphics adapter.

KD_EGA (0x04), for the IBM enhanced graphics adapter.

KIOCSOUND (*)
Start sound generation. Turn on sound. The "arg" is the frequency desired. A frequency of 0 turns off the sound.

KDGETLED Get keyboard LED status. The argument is a pointer to a character. The character will be filled with a boolean combination of the following values:

LED_SCR	0x01	(flag bit for scroll lock)
LED_CAP	0x04	(flag bit for caps lock)
LED_NUM	0x02	(flag bit for num lock)

KDSETLED Set keyboard LED status. The argument is a character whose value is the boolean combination of the values listed under **KDGETLED**.

KDMKTONE (*)
Generate a fixed length tone. The argument is a 32 bit value, with the lower 16 bits set to the frequency and the upper 16 bits set to the duration (in milliseconds).

KDGKBTYPE Get keyboard type. The argument is a pointer to a character type. The character will be returned with one of the following values:

KB_84	0x01	(84 key keyboard)
KB_101	0x02	(101 key keyboard)
KB_OTHER	0x03	

KDADDIO (*)
Add I/O port address to list of valid video adaptor addresses. Argument is an unsigned short type that should contain a valid port address for the installed video adaptor.

KDDELIO (*) Delete I/O port address from list of valid video adaptor addresses. Argument is an unsigned short type that should contain a valid port address for the installed video adaptor.

KDENABIO (*)
Enable in's and out's to video adaptor ports. No argument.

KDDISABIO (*)
Disable in's and out's to video adaptor ports. No argument.

KDQUEMODE (*)
Enable/Disable special queue mode. Queue mode is used by AT&T's X-Windows software to establish a shared queue for access to keyboard and mouse event information. The argument is a pointer to a structure "kd_quemode." If a NULL pointer is sent as an argument, the queue will be closed and the mode disabled. The structure "kd_quemode" is as follows:

```
struct kd_quemode {
      int    qsize;  /* desired # of elements in queue */
      int    signo;  /* signal number to send when queue
                                  goes non-empty */
      void   char    *qaddr; /* user virtual address of queue (set by
                                  driver) */
      };
```

KDSBORDER (*)
Set screen color border in EGA text mode. The argument is of type character. Each bit position corresponds to a color selection. From bit position 0 to bit position 6, the color selections are respectively; blue, green, red, secondary blue, secondary green, and secondary red. Setting the bit position to a logic one will select the desired color or colors. See the NOTES section.

KDSETMODE (*)
Set console in text or graphics mode. The argument is of type integer, which should contain one of the following values:

KD_TEXT	0x00	(sets console to text mode)
KD_GRAPHICS	0x01	(sets console in graphics mode)

If the mode is set to KD_GRAPHICS and the Virtual Terminal is not in process mode (see the VT_SETMODE ioctl), no virtual terminal switches will be possible until the mode is reset to KD_TEXT, KD_TEXT0, or KD_TEXT1.

Note, the user is responsible for programming the color/graphics adaptor registers for the appropriate graphical state.

KDGETMODE (*)
Get current mode of console. Returns integer argument containing either **KD_TEXT** or **KD_GRAPHICS** as defined in the **KDSETMODE ioctl** description.

KDMAPDISP (*)
Maps display memory into user process address space. Argument is a pointer to structure type "kd_memloc." Structure definition is as follows:

```
struct kd_memloc {
     char   *vaddr;        /* virtual address to map to */
     char   *physaddr;     /* physical address to map from */
     long   length;        /* size in bytes to map */
     long   ioflg;         /* enable i/o addresses if set */
     }
```

KDUNMAPDISP (*)
: Unmap display memory from user process address space. No argument required.

KDVDCTYPE This call returns VDC controller/display information.

PIO_FONT8x8 (*)
: This call uses the user supplied 8x8 font.

PIO_FONT8x14 (*)
: This call uses the user supplied 8x14 font.

PIO_FONT8x16 (*)
: This call uses the user supplied 8x16 font.

VT_OPENQRY
: Inquires if this virtual terminal is already open. Find an available virtual terminal. The argument is a pointer to a long. The long will be filled with the number of the first available "VT" that no other process has open or –1 if none is available.

VT_GETMODE (*)
: Determine what mode the active virtual terminal is currently in, either **VT_AUTO** or **VT_PROCESS.** The argument to the **ioctl** is the address of the following type of structure:

```
struct vt_mode {
     char   mode;   /* VT mode */
     char   waitv;  /* if set, hang on writes when not active */
     short  relsig; /* signal to use for release request */
     short  acqsig; /* signal to use for display acquired */
     short  frsig;  /* not used set to 0 */
     }

     #define VT_AUTO 0x00     /* automatic VT switching */
     #define VT_PROCESS     0x01   /* process controls switching */
```

The "vt_mode" structure will be filled in with the current value for each field.

VT_GETSTATE (*)
: The **VT_GETSTATE ioctl** returns global virtual terminal state information. It returns the active virtual terminal in the v_active field, and the number of active virtual terminals and a bit mask of the global state in the v_state field, where bit x is the state of vt x (1 indicates that the virtual terminal is open).

VT_SETMODE (*)
: Set the virtual terminal mode (Auto or Proced). The argument is a pointer to a "vt_mode" structure, as defined above.

VT_SENDSIG (*)
: The **VT_SENDSIG ioctl** specifies a signal (in v_signal) to be sent to a bit mask of virtual terminals (in v_state).

The data structure used by the **VT_GETSTATE** and **VT_SENDSIG ioctl**s is:

```
struct vt_stat {
        ushort v_active;        /* active vt*/
        ushort v_signal;        /* signal to send (VT_SENDSIG) */
        ushort v_state;         /* vt bit mask (VT_SENDSIG and
                                   VT_GETSTATE)*/
};
```

and is defined in /usr/include/sys/vt.h.

VT_RELDISP (*)
: Used to tell the virtual terminal manager that the display has or has not been released by the process. A zero value indicates refusal to release the display. A value of **VT_ACKACQ** indicates an acquisition of a device. EINVAL is returned if a non-zero value that is not equal to VT_ACKACQ is received and the virtual terminal has not yet been acquired. Otherwise, the virtual terminal will be released.

VT_ACTIVATE (*)
: Makes the virtual terminal number specified in the argument the active "VT." The "VT" manager will cause a switch to occur in the same way as if a hotkey sequence had been typed at the keyboard. If the specified "VT" is not open or does not exist, the call will fail and errno will be set to **ENXIO**.

KIOCINFO This call tells the user what the device is.

GIO_SCRNMAP (*)
: This call gets the screen mapping table from the kernel.

GIO_ATTR This call returns the current screen attribute. The bits are interpreted as follows:

Bit 0 determines underlining for black and white monitors (1=underlining on).

Bits 0-2, for color monitors only, select the foreground color. The following list indicates what colors are selected by the given value:

The value 0 selects black.
The value 1 selects red.
The value 2 selects green.
The value 3 selects brown.
The value 4 selects blue.
The value 5 selects magenta.
The value 6 selects cyan.
The value 7 selects white.

Bit 3 is the intensity bit (1=blink on).

Bits 4-6, for color monitors only, select the background color. For a list of colors and their values, see the list under foreground colors.

Bit 7 is the blink bit (1=blink on).

GIO_COLOR (*)
: This call returns a non-zero value if the current display is a color display, otherwise, it returns a zero.

PIO_SCRNMAP
: This call puts the screen mapping table in the kernel.

The screen mapping table maps extended ASCII (8-bit) characters to ROM characters. It is an array [256] of char (typedef *scrnmap_t*) and is indexed by extended ASCII values. The value of the elements of the array are the ROM character to display.

FILES

/dev/console
/dev/vt00-*n*
/dev/video
/usr/include/sys/kd.h

SEE ALSO

console(7), **ioctl**(2), **keyboard**(7), **stty**(1), **termio**(7)

NOTES

Although it is possible to write character sequences that set arbitrary bits on the screen in any of the three graphics modes, this mode of operation is not currently supported.

Monochrome adaptors support underscore option as the default. EGA and VGA adaptors require the use of the **ESC[38m** and **ESC[39m** escape sequences to enable/disable the underscore option respectively. After the underscore option has been enabled on a EGA or VGA adaptor by using the **ESC[38m** sequence and until the underline option has been disabled by using the **ESC[39m** sequence, characters that have blue foreground attributes will be displayed in cyan foreground and characters that have blue background attributes will be displayed in white background attributes.

It is currently not possible to access the 6845 start address registers. Thus, it is impossible to determine the beginning of the color monitor's screen memory.

The alternate/background color bit (bit 4) of the color select register does not appear to affect background colors in alphanumeric modes.

KDSBORDER ioctl calls will not work with AT&T's Super-Vu enhanced color/graphics video adaptor. It will however, work with the IBM EGA card and other EGA compatible video adaptors.

The low-resolution graphics mode appears to be 80 across by 100 down.

NAME

dpt – DPT PM2012 SCSI host adapter subsystem

DESCRIPTION

The **dpt** host adapter subsystem enables SCSI target drivers (such as **sd01**, **st01**, and so on) to communicate on the SCSI bus with target controllers and logical units. This driver implements the Portable Device Interface (PDI) for such PDI target drivers.

It is also possible to access this subsystem directly using the pass-through driver interface. This allows you to issue **sb** control blocks directly to the target controller. To find the appropriate device to use, while any device is being accessed through the target driver (for example, **sd01**), use the **B_GETDEV ioctl** to get the major and minor numbers of the pass-through node. This node may be created and opened for pass-through use (**SDI_SEND ioctl**).

ioctl Calls

The following **ioctl**(2) commands are supported by **dpt**:

SDI_SEND
: Sends a pass-through command (SCSI control block) to a target controller, bypassing the associated target driver.

SDI_BRESET
: Resets the SCSI bus.

B_REDT
: Reads the extended Equipped Device Table (EDT) data structure that is stored in the **dpt** driver's internal data structure.

B_GETTYPE
: Returns the bus name (for example, SCSI) and device driver name of a specific device.

NOTICES

The DPT SCSI adapter will not work reliably with the Emulex SCSI/ESDI bridge controller (also known as the AT&T DCM/4E). In the future, a hardware or software solution may be implemented.

Files

/usr/include/sys/dpt.h
/usr/include/sys/scsi.h
/usr/include/sys/sdi.h
/usr/include/sys/sdi_edt.h
/etc/conf/pack.d/dpt/space.c

REFERENCES

adsc(7), **ioctl**(2), **mcis**(7), **sc01**(7), **sd01**(7), **st01**(7), **sw01**(7), **wd7000**(7)

NAME

ee16 - EtherExpress 16 Ethernet Adapter Driver

SYNOPSIS

```
#include <sys/dlpi.h>
#include <sys/dlpi_ether.h>
#include <sys/ee16.h>

fd = open ("/dev/ee16_0", O_RDWR)
```

DESCRIPTION

The **ee16** driver provides a data link interface to the EtherExpress 16 Ethernet adapter from Intel. It is a STREAMS-based driver that is compatible with the Data Link Provider Interface (DLPI) and Logical Link Interface (LLI) software interfaces.

It supports **DL_ETHER** and **DL_CSMACD** for MAC type, **DL_CL_ETHER** for service mode, and **DL_STYLE1** for provider style. The **ee16** driver can operate as a cloned or non-cloned device.

A process must issue a **DL_BIND_REQ** primitive to receive frames from the network. The process must specify the *dl_sap* field of the **dl_bind_req_t** structure. The *type* field of an incoming frame is compared to the *dl_sap* value. If the values are equal, it is placed on the STREAMS read queue of the process. A privileged process may set the *dl_sap* field to **PROMISCUOUS_SAP**. The **PROMISCUOUS_SAP** matches all incoming frames.

A privileged process may also bind to a SAP already bound by another process. In cases where a frame qualifies to be sent to more than one process, independent copies of the frame will be made and placed on the STREAMS read queue of each process.

Received frames are delivered in **dl_unitdata_ind_t** structures. The source and destination address each contain a 6-byte Ethernet address, followed by a 2-byte type value.

USAGE

ioctl Calls

The following **ioctls** are supported:

DLIOCGMIB

Returns the **DL_mib_t** structure, which contains the Management Information Base (MIB). The MIB holds the Ethernet statistics kept in the driver.

```
/*
 *  Ether statistics structure.
 */
typedef struct {
  ulong_t    etherAlignErrors;        /* Frame alignment errors */
  ulong_t    etherCRCerrors;          /* CRC erros */
  ulong_t    etherMissedPkts;         /* Packet overflow or missed inter */
  ulong_t    etherOverrunErrors;      /* Overrun errors */
  ulong_t    etherUnderrunErrors;     /* Underrun errors */
  ulong_t    etherCollisions;         /* Total collisions */
  ulong_t    etherAbortErrors;        /* Transmits aborted at interface  */
  ulong_t    etherCarrierLost;        /* Carrier sense signal lost */
  ulong_t    etherReadqFull;          /* STREAMS read queue full */
  ulong_t    etherRcvResources;       /* Receive resource alloc faliure  */
  ulong_t    etherDependent1;         /* Device dependent statistic */
  ulong_t    etherDependent2;         /* Device dependent statistic */
  ulong_t    etherDependent3;         /* Device dependent statistic */
  ulong_t    etherDependent4;         /* Device dependent statistic */
  ulong_t    etherDependent5;         /* Device dependent statistic */
} DL_etherstat_t;

/*
 *  Interface statistics compatible with MIB II SNMP requirements.
 */
typedef struct {
  int        ifIndex;                 /* ranges between 1 and ifNumber */
  int        ifDescrLen;              /* len of desc. following this struct */
  int        ifType;                  /* type of interface */
  int        ifMtu;                   /* datagram size that can be sent/rcv */
  ulong_t    ifSpeed;                 /* estimate of bandwidth in bits PS */
  uchar_t    ifPhyAddress[DL_MAC_ADDR_LEN];/* Ethernet Address */
  int        ifAdminStatus;           /* desired state of the interface */
  int        ifOperStatus;            /* current state of the interface */
  ulong_t    ifLastChange;            /* sysUpTime when state was entered */
  ulong_t    ifInOctets;              /* octets received on interface */
  ulong_t    ifInUcastPkts;           /* unicast packets delivered */
  ulong_t    ifInNUcastPkts;          /* non-unicast packets delivered */
  ulong_t    ifInDiscards;            /* good packets received but dropped */
  ulong_t    ifInErrors;              /* packets received with errors */
  ulong_t    ifInUnknownProtos;       /* packets recv'd to unbound proto */
  ulong_t    ifOutOctets;             /* octets transmitted on interface */
  ulong_t    ifOutUcastPkts;          /* unicast packets transmited */
  ulong_t    ifOutNUcastPkts;         /* non-unicast packets transmited */
  ulong_t    ifOutDiscards;           /* good outbound packets dropped */
  ulong_t    ifOutErrors;             /* number of transmit errors */
  ulong_t    ifOutQlen;               /* length of output queue */
  DL_etherstat_t ifSpecific;          /* ethernet specific stats */
} DL_mib_t;
```

The values in the MIB are compatible with those needed by the SNMP protocol.

The *ifDescrLen* field indicates the length of the null terminated description string that immediately follows the `DL_mib_t` structure.

There are three fields in the MIB that are specific to the **ee16** driver. The *ifSpecific.etherDependent1* field tracks the number of times the transceiver failed to transmit a collision signal after transmission of a packet. The *ifSpecific.etherDependent2* field contains the number of collisions that occurred after a slot time (out of window collisions). The *ifSpecific.etherDependent3* field tracks the number of times a transmit interrupt timeout condition occurred.

DLIOCSMIG
: Allows a privileged process to initialize the values in the MIB (that is, the **DL_mib_t** structure). A process cannot use this **ioctl** to change the *ifPhyAddress*, the *ifDescrLen*, or the text of the description fields.

DLIOCGENADDR
: Returns the Ethernet address in network order.

DLIOCGLPCFLG
: Returns the state of the local packet copy flag in the *ioc_rval* of the **iocblk** structure. The local copy flag determines if packets looped back by the driver should also be sent to the network. A non-zero value indicates that frames should also be be sent to the network after being looped back. The default value of this flag is zero.

DLIOCSLPCFLG
: Allows a privileged process to set the local packet copy flag, causing all packets looped back by the driver to be sent to the network as well.

DLIOCGPROMISC
: Returns the value of the promiscuous flag in the *ioc_rval* of the **iocblk** structure. A non-zero value indicates that the Ethernet interface will receive all frames on the network. The default value of this flag is zero.

DLIOCSPROMISC
: Allows a privileged process to toggle the current state of the promiscuous flag. When the flag is set, the driver captures all frames from the network. Processes that are bound to a promiscuous SAP, or to a SAP that matches the type field of the received frame, receive a copy of the frame.

DLIOCGETMULTI
: Returns a list of multicast addresses (if it exists).

DLIOCADDMULTI
: Allows a privileged process to add a new multicast address and enable its reception. A 6-byte buffer pointing to the multicast address must be passed as the parameter.

DLIOCDELMULTI
: Allows a privileged process to delete a multicast address by passing a 6-byte multicast address as the parameter.

Installation

You must select the interrupt level and the base I/O address for the EtherExpress card at package installation time. Refer to the User's Guide that came with your adaptor for more information.

You can also set these parameters in the **/etc/conf/sdevice.d/ee16** file.

If there is one EtherExpress 16 card in the system, it is configured automatically to the parameters specified at installation each time the system is initialized. During initialization, the system determines the current base I/O address of the card, and then programs the card to its new configuration. If the system doesn't find an EtherExpress 16 card, it displays an error message. Because of the way the system searches for an EtherExpress 16 card, reads to I/O addresses without EtherExpress 16 cards may occur during the automatic configuration process.

Automatic configuration occurs only when one EtherExpress 16 card is installed in the system. If you want to install more than one card, use the following procedure:

When you install the `ee16` package, specify the interrupt level and the base I/O address for one EtherExpress 16 card, install the card in the system, reboot the system, and use the automatic configuration process to assign those parameters to the card.

Then, for each additional card, edit the `/etc/conf/sdevice.d/ee16` file and specify the interrupt level and base I/O address for the next EtherExpress 16 card (being careful not to use an interrupt level or base I/O address that are being used by any other board in the system), shut down the system, remove the EtherExpress 16 card that's in the system, replace it with the next EtherExpress 16 card, and reboot the system. The automatic configuration process will assign the parameters in `/etc/conf/sdevice.d/ee16` to the new card.

When you have programmed the parameters for all the cards, shut down and turn off the system, and install all the cards. The configuration files need to reflect the fact that multiple cards are in use. When you boot the system it recognizes that there are multiple EtherExpress 16 cards installed and does not try to autoconfigure them.

Configuration

The `ee16` driver has four configurable parameters in the `/etc/conf/pack.d/ee16/space.c` file. Any changes to this file must be followed by a rebuild of the kernel and a reboot of the system for the changes to take effect.

The configurable parameters are:

`N_SAPS`
Defines the number of SAPs that can be bound at any one time. This value should be only slightly larger than anticipated SAP usage. A typical TCP/IP system requires two SAPs (0x800 and 0x806). A large value will degrade performance and increase memory usage.

`CABLE_TYPE`
Defines the type of Ethernet cable attached to the Ethernet controller card. A value of 0 specifies a thin Ethernet cable with a BNC connector. A value of 1 specifies a thick Ethernet cable with an AUI connector.

`STREAMS_LOG`
Defines whether the driver should log debugging messages to the STREAMS logger for the **strace**(1M) utility to display. The module ID used with **strace** is 2101. A value of 0 indicates that no STREAMS debug messages should be generated. A value of 1 will cause messages to be generated. You can temporarily set the driver to log messages by changing the value of *ee16strlog* (a 4-byte integer) to 1 using the kernel debugger.

STREAMS tracing should only be performed when debugging a network problem. It can cause a severe performance degradation if you use full `ee16` STREAMS logging.

IFNAME
: This parameter is important only in a TCP/IP networking environment. It defines the string used in displaying network statistics. This string should match the logical interface name assigned in **/etc/confnet.d/inet/interfaces** file and with **ifconfig**(1M) commands used in **/etc/inet/rc.inet** configuration script.

Errors

The **ee16** driver can return the following error codes:

ENXIO Invalid major number or board is not installed.

ECHRNG
No minor devices left if configured as a cloned device. Increase **N_SAP** value in **/etc/conf/pack.d/ee16/space.c** Invalid minor device number if configured as a non-cloned device.

EPERM An **ioctl** was made without the appropriate privilege.

EINVAL
An **ioctl** was made that did not supply a required input and/or output buffer.

DL_NOTSUPPORTED
Requested service primitive is not supported.

DL_BADPRIM
Unknown service primitive was requested.

DL_OUTSTATE
DL_BIND_REQ was issued when the Stream was bound, or **DL_UNBIND_REQ** or **DL_UNITDATA_REQ** was issued when the Stream was unbound.

DL_ACCESS
An attempt was made to bind to **PROMISCUOUS_SAP** with insufficient privilege.

DL_BOUND
The requested SAP is already bound. A privileged process may bind to an already bound SAP.

DL_NOTINIT
DL_UNITDATA_REQ was made on an Ethernet board that has gone offline due to an error.

DL_BADDATA
DL_UNITDATA_REQ was made with a data size that was either larger than the SPDU maximum or smaller than the SPDU minimum.

Files

/dev/ee16*
/etc/conf/pack.d/ee16/space.c
/etc/conf/sdevice.d/ee16

REFERENCES

getmsg(2), **ioctl**(2), **open**(2), **putmsg**(2)

NAME

el16 - EtherLink 16 Ethernet Adapter Driver

SYNOPSIS

```
#include <sys/dlpi.h>
#include <sys/dlpi_ether.h>
#include <sys/el16.h>

fd = open ("/dev/el16_0", O_RDWR)
```

DESCRIPTION

The **el16** driver provides a data link interface to the EtherLink 16 and Etherlink/MC adapters from 3Com. It is a STREAMS-based driver compatible with the Data Link Provider Interface (DLPI) and Logical Link Interface (LLI) software interfaces.

It supports **DL_ETHER** and **DL_CSMACD** for MAC type, DL_CLDLS for service mode, and DL_STYLE1 for provider style. The **el16** driver can operate as a cloned or non-cloned device.

A process must issue a **DL_BIND_REQ** primitive to receive frames from the network. The process must specify the *dl_sap* field of the **dl_bind_req_t** structure. The type field of an incoming frame is compared to the *dl_sap* value. If the values are equal, it is placed on the STREAMS read queue of the process. A privileged process may set the *dl_sap* field to **PROMISCUOUS_SAP**. The **PROMISCUOUS_SAP** matches all incoming frames.

A privileged process may also bind to a SAP already bound by another process. In cases where a frame qualifies to be sent to more than one process, independent copies of the frame are created and placed on the STREAMS read queue of each process.

Received frames are delivered in **dl_unitdata_ind_t** structures. The source and destination address each contain contain a 6-byte Ethernet address, followed by a 2-byte type value.

USAGE

ioctl Calls

The following **ioctl**s are supported:

DLIOCGMIB

Returns the **DL_mib_t** structure, which contains the Management Information Base (MIB). The MIB holds the Ethernet statistics kept in the driver.

```
/*
 *  Ether statistics structure.
 */
typedef struct {
  ulong_t    etherAlignErrors;        /* Frame alignment errors */
  ulong_t    etherCRCerrors;          /* CRC erros */
  ulong_t    etherMissedPkts;         /* Packet overflow or missed inter */
  ulong_t    etherOverrunErrors;      /* Overrun errors */
  ulong_t    etherUnderrunErrors;     /* Underrun errors */
  ulong_t    etherCollisions;         /* Total collisions */
```

```
   ulong_t  etherAbortErrors;       /* Transmits aborted at interface  */
  ulong_t   etherCarrierLost;       /* Carrier sense signal lost */
  ulong_t   etherReadqFull;         /* STREAMS read queue full */
  ulong_t   etherRcvResources;      /* Receive resource alloc faliure  */
  ulong_t   etherDependent1;        /* Device dependent statistic */
  ulong_t   etherDependent2;        /* Device dependent statistic */
  ulong_t   etherDependent3;        /* Device dependent statistic */
  ulong_t   etherDependent4;        /* Device dependent statistic */
  ulong_t   etherDependent5;        /* Device dependent statistic */
} DL_etherstat_t;

/*
 *  Interface statistics compatible with MIB II SNMP requirements.
 */
typedef struct {
  int       ifIndex;                /* ranges between 1 and ifNumber */
  int       ifDescrLen;             /* len of desc. following this struct */
  int       ifType;                 /* type of interface */
  int       ifMtu;                  /* datagram size that can be sent/rcv */
  ulong_t   ifSpeed;                /* estimate of bandwidth in bits PS */
  uchar_t   ifPhyAddress[DL_MAC_ADDR_LEN];/* Ethernet Address */
  int       ifAdminStatus;          /* desired state of the interface */
  int       ifOperStatus;           /* current state of the interface */
  ulong_t   ifLastChange;           /* sysUpTime when state was entered */
  ulong_t   ifInOctets;             /* octets received on interface */
  ulong_t   ifInUcastPkts;          /* unicast packets delivered */
  ulong_t   ifInNUcastPkts;         /* non-unicast packets delivered */
  ulong_t   ifInDiscards;           /* good packets received but dropped */
  ulong_t   ifInErrors;             /* packets received with errors */
  ulong_t   ifInUnknownProtos;      /* packets recv'd to unbound proto */
  ulong_t   ifOutOctets;            /* octets transmitted on interface */
  ulong_t   ifOutUcastPkts;         /* unicast packets transmited */
  ulong_t   ifOutNUcastPkts;        /* non-unicast packets transmited */
  ulong_t   ifOutDiscards;          /* good outbound packets dropped */
  ulong_t   ifOutErrors;            /* number of transmit errors */
  ulong_t   ifOutQlen;              /* length of output queue */
  DL_etherstat_t ifSpecific;        /* ethernet specific stats */
} DL_mib_t;
```

The values in the MIB are compatible with those needed by the SNMP protocol.

The *ifDescrLen* field indicates the length of the null terminated description string that immediately follows the *DL_mib_t* structure.

There are three fields in the MIB that are specific to the **el16** driver. The *ifSpecific.etherDependent1* field tracks the number of times the transceiver failed to transmit a collision signal after transmission of a packet. The *ifSpecific.etherDependent2* field contains the number of collisions that occurred after a slot time (out of window collisions). The *ifSpecific.etherDependent3* field tracks the number of times a transmit interrupt timeout condition occurred.

DLIOCSMIB
Allows a privileged process to initialize the values in the MIB (that is, the **DL_mib_t** structure). A process cannot use this **ioctl** to change the *ifPhyAddress*, the *ifDescrLen*, or the text of the description fields.

DLIOCGENADDR
Returns the Ethernet address in network order.

DLIOCGLPCFLG
Returns the state of the local packet copy flag in the *ioc_rval* of the **iocblk** structure. The local copy flag determines if packets looped back by the driver should also be sent to the network. A non-zero value indicates that frames should also be be sent to the network after being looped back. The default value of this flag is zero.

DLIOCSLPCFLG
Allows a privileged process to set the local packet copy flag, causing all packets looped back by the driver to be sent to the network as well.

DLIOCGPROMISC
Returns the value of the promiscuous flag in the *ioc_rval* of the **iocblk** structure. A non-zero value indicates that the Ethernet interface will receive all frames on the network. The default value of this flag is zero.

DLIOCSPROMISC
Allows a privileged process to toggle the current state of the promiscuous flag. When the flag is set, the driver captures all frames from the network. Processes that are bound to a promiscuous SAP, or to a SAP that matches the type field of the received frame, receive a copy of the frame.

DLIOCGETMULTI
Retries the current list of multicast addresses (if it exists).

DLIOCADDMULTI
Allows a privileged process to add a new multicast address and enable its reception. A 6-byte buffer pointing to the multicast address must be passed as the parameter.

DLIOCDELMULTI
Allows a privileged process to delete a multicast address by passing a 6-byte multicast address as the parameter.

Installation

You can select the interrupt vector, the base I/O address, and the base RAM address for the EtherLink 16 adaptor at package installation time. Consult the User's Manual that came with your adaptor for more information.

You can also set these parameters in the **/etc/conf/sdevice.d/el16** file.

In addition, zero-wait-state operation of memory can be enabled or disabled at installation time.

If there is one EtherLink 16 card in the system, it is configured automatically to the parameters specified at installation when the system is initialized. During initialization, the system determines the current base I/O address of the card, and then programs the card to its new configuration. If the system doesn't find an EtherLink 16 card, it displays an error message.

Automatic configuration occurs only when one EtherLink 16 card is installed in the system. If more than one EtherLink 16 card needs to be present in the system, the cards are assumed to have been configured already (the cards may be configured using the configuration software that comes with the card).

Configuration

The `el16` driver has six configurable parameters in the `/etc/conf/pack.d/el16/space.c` file. Any changes to this file must be followed by a rebuild of the kernel and a reboot of the system for the changes to take effect.

The configurable parameters are:

`N_SAPS`
: This defines the number of SAPs that can be bound at any one time. This value should be only slightly larger than anticipated SAP usage. A typical TCP/IP system would require two SAPs (0x800 and 0x806). A large value will degrade system performance and increase memory usage.

`CABLE_TYPE`
: Defines the type of ethernet cable attached to the Ethernet controller card. A value of 0 specifies a thin Ethernet cable with a BNC connector. A value of 1 specifies a thick Ethernet cable with an AUI connector.

`STREAMS_LOG`
: Defines whether the driver should log debugging messages to the STREAMS logger for the **`strace`**(1M) utility to display. The module ID used with **`strace`** is 2101. A value of 0 indicates that no STREAMS debug messages should be generated. A value of 1 causes messages to be generated. You can temporarily instruct the driver to log messages by changing the value of **`el16strlog`** (a 4-byte integer) to 1 using the kernel debugger.

: STREAMS tracing should only be performed when debugging a network problem. It can cause a severe performance degradation if full **`el16`** STREAMS logging is performed.

`IFNAME`
: This parameter is important only in a TCP/IP networking environment. It defines the string used in displaying network statistics. This string should match the logical interface name assigned in `/etc/confnet.d/inet/interface` file and with `ifconfig`(1M) commands used in `/etc/inet/rc.inet` configuration script.

`ZWS` Enables or disables zero-wait-state operation of the memory. A value of 0 disables zero-wait-state operation while a value of 1 enables it.

`SAAD` Enables or disables software address decode in the adapter. In machines which use the VTI chip set, this bit needs to be set.

Errors

The **el16** driver can return the following error codes:

ENXIO Invalid major number or board is not installed.

ECHRNG
No minor devices left if configured as a cloned device. Increase **N_SAP** value in **/etc/conf/pack.d/el16/space.c** Invalid minor device number if configured as a non-cloned device.

EPERM An **ioctl** was made without the appropriate privilege.

EINVAL
An **ioctl** was made that did not supply a required input and/or output buffer.

DL_NOTSUPPORTED
Requested service primitive is not supported.

DL_BADPRIM
Unknown service primitive was requested.

DL_OUTSTATE
DL_BIND_REQ or **DL_UNBIND_REQ** was issued when the Stream was bound, or **DL_UNITDATA_REQ** was issued when the Stream was unbound.

DL_ACCESS
An attempt was made to bind to **PROMISCUOUS_SAP** with insufficient privilege.

DL_BOUND
The requested SAP is already bound. A privileged process may bind to an already bound SAP.

DL_NOTINIT
A **DL_UNITDATA_REQ** was made on an Ethernet board that has gone offline due to an error.

DL_BADDATA
DL_UNITDATA_REQ was made with a data size that was either larger than the SPDU maximum or smaller than the SPDU minimum.

Files

/dev/el16*
/etc/conf/pack.d/el16/space.c
/etc/conf/sdevice.d/el16

REFERENCES

open(2), **getmsg**(2), **ioctl**(2), **putmsg**(2)

NAME

`fd` – diskette (floppy disk)

DESCRIPTION

The diskette driver provides access to diskettes as both block and character devices. Diskettes must be formatted before their use [see **`format`**(1M)]. Both 5.25" and 3.50" diskette formats are supported. The driver controls up to two diskette drives. The minor device number specifies the drive number, the format of the diskette and the partition number.

Diskette device file names (which correspond to a specific major and minor device) use the following format:

```
/dev/{r}dsk/f{0,1}{5h,5d9,5d8,5d4,5d16,5q,3h,3d}{t,u}
```

where **`r`** indicates a raw (character) interface to the diskette, **`rdsk`** selects the raw device interface and **`dsk`** selects the block device interface. **`0`** or **`1`** selects the drive to be accessed: **`f0`** selects floppy drive 0, while **`f1`** selects drive 1. The following list describes the format to be interacted with:

`5h`	5.25" high density diskette (1.2MB).
`5d9`	5.25" double density diskette, 9 sectors per track (360KB).
`5d8`	5.25" double density diskette, 8 sectors per track (320KB).
`5d4`	5.25" double density diskette, 4 sectors per track (320KB).
`5d16`	5.25" double density diskette, 16 sectors per track (320KB).
`5q`	5.25" quad density diskette (720KB).
`3h`	3.50" high density diskette (1.44MB).
`3d`	3.50" double density diskette (720KB).

Format specification is mandatory when opening the device for formatting. However, when accessing a floppy disk for other operations (read and write), the format specification field can be omitted. In this case, the floppy disk driver will automatically determine the format previously established on the diskette and then perform the requested operation (for example, **`cpio -itv</dev/rsdk/f1)`**.

The last parameter, **`t`** or **`u`**, selects the partition to be accessed. **`t`** represents the whole diskette. Without **`t`** or **`u`** specified, the whole diskette except cylinder 0 will be selected. **`u`** represents the whole diskette except track 0 of cylinder 0 and applies only to the 5d8 type of floppy.

Besides the device file naming convention described above, some of the formats have alias names that correlate to previous releases. The following list describes the formats that have an alias:

format	alias
`5h`	q15d
`5d8`	**`d8d`**
`5d9`	**`d9d`**

For example, the device file **`/dev/rdsk/f0q15dt`** is equivalent to **`/dev/rdsk/f05ht`**.

In order to minimize errors when using diskettes, the driver attempts to assure that the diskette is installed when needed, and that the operations requested have been completed before the device close is completed. In particular, the drive is checked for the presence of a diskette each time a read/write request is made to the drive. If this is not true (either the diskette is not physically present or the door is open), the driver retries the request continually, at five-second intervals. The message:

```
FD(n): diskette not present - please insert
```

appears after each attempt (the *n* represents the drive number). The INTR and QUIT signals are honored in this case, so that the process accessing the diskette drive in question will receive these signals (unless, of course, the process itself is ignoring them). In particular, if the diskette is removed prematurely, or not inserted soon enough, *no data is lost,* provided the correct diskette is inserted in the drive when the message to do so is displayed.

ioctl Calls

V_GETPARMS

This call is used to get information about the current drive configuration. The argument to the **ioctl** is the address of one of the following structures, defined in **sys/vtoc.h**, which will be filled in by the **ioctl**:

```
struct disk_parms {
        char   dp_type;         /* Disk type (see below) */
        unchar dp_heads;        /* Number of heads */
        ushort dp_cyls;         /* Number of cylinders */
        unchar dp_sectors;      /* Number of sectors/track */
        ushort dp_secsiz;       /* Number of bytes/sector */
                                /* for this partition: */
        ushort dp_ptag;         /* Partition tag (not used) */
        ushort dp_pflag;        /* Partition flag (not used) */
        ushort dp_pstartsec;    /* Starting sector number */
        ushort dp_pnumsec;      /* Number of sectors */
}

/* Disk types */
#defineDPT_WINI       1         /* Winchester disk */
#defineDPT_FLOPPY     2         /* Floppy */
#defineDPT_OTHER      3         /* Other type of disk */
#defineDPT_NOTDISK    0         /* Not a disk device */
```

For the floppy driver, the disk type will always be DPT_FLOPPY. The unused fields in the disk_parms structure are only applicable to hard disks; however, returning the same structure from both the hard disk driver and the diskette driver allows programs to be written that can understand either one.

V_FORMAT

This call is used to format tracks on a diskette. The argument passed to the **ioctl** is the address of one of the following structures, defined in **sys/vtoc.h**, containing the starting track, number of tracks, and interleave factor:

```
union io_arg {
        struct {
                ushort start_trk;     /* first track */
                ushort num_trks;      /* number of tracks
                                      to format */
                ushort intlv;         /* interleave factor */
        } ia_fmt;
}
```

Formatting will start at the given track and will continue so that the given number of tracks are formatted, using the given interleave factor.

Note that the file descriptor must refer to the character (raw) special device for the desired drive, and the file must have been opened in exclusive mode (that is, O_EXCL).

Files

```
/dev/dsk/f0, /dev/rdsk/f0, ...
/dev/dsk/f0t, /dev/rdsk/f0t, ...
/dev/dsk/f05h, /dev/rdsk/f05h, ...
/dev/dsk/f05ht, /dev/rdsk/f05ht, ...
/dev/dsk/f05d9, /dev/rdsk/f05d9, ...
/dev/dsk/f05d9t, /dev/rdsk/f05d9t, ...
/dev/dsk/f0fd8, /dev/rdsk/f05d8, ...
/dev/dsk/f05d8t, /dev/rdsk/f05d8t, ...
/dev/dsk/f05d4, /dev/rdsk/f05d4, ...
/dev/dsk/f05d4t, /dev/rdsk/f05d4t, ...
/dev/dsk/f05d16, /dev/rdsk/f05d16, ...
/dev/dsk/f05d16t, /dev/rdsk/f05d16t, ...
/dev/dsk/f05q, /dev/rdsk/f05q, ...
/dev/dsk/f05qt, /dev/rdsk/f05qt, ...
/dev/dsk/f03h, /dev/rdsk/f03h, ...
/dev/dsk/f03ht, /dev/rdsk/f03ht, ...
/dev/dsk/f03d, /dev/rdsk/f03d, ...
/dev/dsk/f03dt, /dev/rdsk/f03dt, ...
```

Errors

The driver will retry failed transfers up to ten times. If the request still has not succeeded, the driver will display an appropriate message. Errors from the diskette controller, other than the above, are displayed as follows:

```
FD drv n, blk b: drive error message
FD controller controller error message
```

The first message occurs on an error after a transfer has begun, where **n** is the drive where the error occurred, and **b** is the block number that is being read or written. The *drive error message* is one of the messages appearing in the following list:

Missing data address mark
: The diskette may not be formatted properly.

`Cylinder marked bad`
: The accessed cylinder has been marked bad by the formatter.

`Seek error (wrong cylinder)`
: The drive positioned itself at the wrong cylinder when attempting to set up for the requested transfer.

`Uncorrectable data read error`
: A CRC error was detected when attempting to read the requested block from the drive.

`Sector marked bad`
: The accessed sector has been marked bad by the formatter.

`Missing header address mark`
: The diskette may not be formatted properly.

`Write protected`
: A write was attempted to a diskette that is currently write-protected.

`Sector not found`
: The diskette may not be formatted properly.

`Data overrun`
: The system could not keep up with the requested transfer of data. (Should not occur.)

`Header read error`
: The diskette may not be formatted properly.

`Illegal sector specified`
: The driver is confused about the format of the diskette that has been inserted. (Should not occur.)

The second message occurs when there is a controller error during the setup for, or actual transfer of, a block. The *controller error message* is one of the messages appearing in the following list:

`command timeout`
: The controller failed to complete the requested command in a reasonable length of time.

`status timeout`
: The controller failed to return its status after a command was completed.

`busy` During an attempt to access the controller, a timeout occurred.

REFERENCES

`format`(1M), `ioctl`(2)

NAME

`filesystem` – file system organization

SYNOPSIS

`/`

`/usr`

DESCRIPTION

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

The file system tree consists of a root file system and a collection of mountable file systems. The **`mount`**(1M) program attaches mountable file systems to the file system tree at mount points (directory entries) in the root file system or other previously mounted file systems. `/` If /f4/usr is configured as a separate file system, it must be mounted in order to have a completely functional system. The root file system is mounted automatically by the kernel at boot time.

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

`/dev`	Character and block special files. These device files provide hooks into hardware devices or operating system facilities. Typically, device files are built to match the kernel and hardware configuration of the machine.
`/dev/term`	Terminal devices.
`/dev/pts`	Pseudo-terminal devices.
`/dev/sxt`	Shell layers device files used by `shl.`
`/etc`	Machine-specific administrative configuration files and system administration databases. `/etc` may be viewed as the home directory of a machine, the directory that in a sense defines the machine's identity. Executable programs are no longer kept in `/etc`.
`/home`	Root of a subtree for user directories.
`/mnt`	Temporary mount point for file systems. This is an empty directory on which file systems may be temporarily mounted.
`/opt`	Root of a subtree for add-on application packages.
`/proc`	Root of a subtree for the process file system.
`/sbin`	Essential executables used in the booting process and in manual system recovery. The full complement of utilities is available only after `/usr` is mounted,

/tmp	Temporary files; initialized to empty during the boot operation.
/var	Root of a subtree for varying files. Varying files are files that are unique to a machine but that can grow to an arbitrary (that is, variable) size. An example is a log file.
/var/adm	System logging and accounting files.
/var/cron	**cron**'s log file.
/var/mail	Where users' mail is kept.
/var/opt	Top-level directory used by application packages.
/var/preserve	Backup files for **vi**(1) and **ex**(1).
/var/spool	Subdirectories for files used in printer spooling, mail delivery, **cron**(1M), **at**(1), etc.
/var/tmp	Transitory files; initialized to empty during the boot operation.

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

The file system mounted on **/usr** contains architecture-dependent and architecture-independent sharable files. The subtree rooted at **/usr/share** contains architecture-independent sharable files; the rest of the **/usr** tree contains architecture-dependent files. By mounting a common remote file system, a group of machines with a common architecture may share a single **/usr** file system. A single **/usr/share** file system can be shared by machines of any architecture. A machine acting as a file server may export many different **/usr** file systems to support several different architectures and operating system releases. Clients usually mount **/usr** read-only so that they don't accidentally change any shared files. The **/usr** file system contains the following subdirectories:

/usr/bin	Most system utilities.
/usr/sbin	Executables for system administration.
/usr/games	Game binaries and data.
/usr/include	Include header files (for C programs, etc).
/usr/lib	Program libraries, various architecture-dependent databases, and executables not invoked directly by the user (system daemons, etc).
/usr/share	Subtree for architecture-independent sharable files.
/usr/share/man	Subdirectories for on-line reference manual pages (if present).
/usr/share/lib	Architecture-independent databases.
/usr/src	Source code for utilities and libraries.
/usr/ucb	Berkeley compatibility package binaries.

/usr/ucbinclude	Berkeley compatibility package header files.
/usr/ucblib	Berkeley compatibility package libraries.

A machine with disks may export root file systems, swap files, and **/usr** file systems to diskless or partially-disked machines that mount them into the standard file system hierarchy. The standard directory tree for sharing these file systems from a server is:

/export	The default root of the exported file system tree.
/export/exec/*architecture-name*	The exported **/usr** file system supporting *architecture-name* for the current release.
/export/exec/*architecture-name***.***release-name*	The exported **/usr** file system supporting *architecture-name* for System V *release-name*.
/export/exec/share	The exported common **/usr/share** directory tree.
/export/exec/share.*release-name*	The exported common **/usr/share** directory tree for System V *release-name*.
/export/root/*hostname*	The exported root file system for *hostname*.
/export/swap/*hostname*	The exported swap file for *hostname*.
/export/var/*hostname*	The exported **/var** directory tree for *hostname*.

SEE ALSO

at(1), **fsck**(1M), **init**(1M), **mknod**(1M), **mount**(1M), **sh**(1), **vi**(1)

NAME

`i596` – i596 Ethernet Driver

SYNOPSIS

```
#include <sys/dlpi.h>
#include <sys/dlpi_ether.h>
#include <sys/i596.h>

fd = open ("/dev/i596_0", O_RDWR)
```

DESCRIPTION

The **i596** driver provides a data link interface to the 82596 high-performance 32-bit LAN coprocessor in the LP486/33E system. In this system, the 82596CA resides on the host bus, sharing address, data, and control lines with the i486 processor. This driver is a STREAMS-based driver that is compatible with the Data Link Provider Interface (DLPI) and Logical Link Interface (LLI) software interfaces.

It supports **DL_ETHER** as MAC type, **DL_CL_ETHER** for service mode, and **DL_STYLE1** for provider style. The **i596** driver can operate as a cloned or non-cloned device.

A process must issue a **DL_BIND_REQ** primitive to receive frames from the network. The process must specify the *dl_sap* field of the **dl_bind_req_t** structure. The *type* field of an incoming frame is compared to the *dl_sap* value. If the values are equal, it is placed on the STREAMS read queue of the process. A privileged process may set the *dl_sap* field to **PROMISCUOUS_SAP**. The **PROMISCUOUS_SAP** matches all incoming frames.

A privileged process may also bind to a SAP already bound by another process. In cases where a frame qualifies to be sent to more than one process, independent copies of the frame are created and placed on the STREAMS read queue of each process.

Received frames are delivered in **dl_unitdata_ind_t** structures. The source and destination address each contain a 6-byte Ethernet address followed by a 2-byte type value.

USAGE

ioctl Calls

The following **ioctl**s are supported:

DLIOCGMIB

Returns the **DL_mib_t** structure, which contains the Management Information Base (MIB). The MIB holds the Ethernet statistics that are kept in the driver.

```
/*
 *  Ether statistics structure.
 */
typedef struct {
  int       etherAlignErrors;       /* Frame alignment errors */
  int       etherCRCerrors;         /* CRC errors */
  int       etherMissedPkts;        /* Packet overflow or missed inter */
  int       etherOverrunErrors;     /* Overrun errors */
  int       etherUnderrunErrors;    /* Underrun errors */
```

```
   int       etherCollisions;        /* Total collisions */
  int       etherAbortErrors;       /* Transmits aborted at interface  */
  int       etherCarrierLost;       /* Carrier sense signal lost */
  int       etherReadqFull;         /* STREAMS read queue full */
  int       etherRcvResources;      /* Receive resource alloc failure  */
  int       etherDependent1;        /* Device dependent statistic */
  int       etherDependent2;        /* Device dependent statistic */
  int       etherDependent3;        /* Device dependent statistic */
  int       etherDependent4;        /* Device dependent statistic */
  int       etherDependent5;        /* Device dependent statistic */
} DL_etherstat_t;

/*
 *  Interface statistics compatible with MIB II SNMP requirements.
 */
typedef struct {
  int       ifIndex;                /* ranges between 1 and ifNumber */
  int       ifDescrLen;             /* len of desc. following this struct */
  int       ifType;                 /* type of interface */
  int       ifMtu;                  /* datagram size that can be sent/rcv */
  ulong_t   ifSpeed;                /* estimate of bandwidth in bits PS */
  uchar_t   ifPhyAddress[DL_MAC_ADDR_LEN];/* Ethernet Address */
  int       ifAdminStatus;          /* desired state of the interface */
  int       ifOperStatus;           /* current state of the interface */
  ulong_t   ifLastChange;           /* sysUpTime when state was entered */
  ulong_t   ifInOctets;             /* octets received on interface */
  ulong_t   ifInUcastPkts;          /* unicast packets delivered */
  ulong_t   ifInNUcastPkts;         /* non-unicast packets delivered */
  ulong_t   ifInDiscards;           /* good packets received but dropped */
  ulong_t   ifInErrors;             /* packets received with errors */
  ulong_t   ifInUnknownProtos;      /* packets recv'd to unbound proto */
  ulong_t   ifOutOctets;            /* octets transmitted on interface */
  ulong_t   ifOutUcastPkts;         /* unicast packets transmited */
  ulong_t   ifOutNUcastPkts;        /* non-unicast packets transmited */
  ulong_t   ifOutDiscards;          /* good outbound packets dropped */
  ulong_t   ifOutErrors;            /* number of transmit errors */
  ulong_t   ifOutQlen;              /* length of output queue */
  DL_etherstat_t ifSpecific;        /* ethernet specific stats */
} DL_mib_t;
```

The values in the MIB are compatible with those needed by the SNMP protocol.

The *ifDescrLen* field indicates the length of the null terminated description string that immediately follows the `DL_mib_t` structure.

There are three fields in the MIB that are specific to the `i596` driver: The *ifSpecific.etherDependent1* field tracks the number of times the transceiver failed to transmit a collision signal after transmission of a packet. The *ifSpecific.etherDependent2* field contains the number of collisions that occurred after a slot time (out of window collisions). The *ifSpecific.etherDependent3* field tracks the number of times a transmit interrupt timeout condition occurred.

DLIOCSMIG
: Allows a privileged process to initialize the values in the MIB (that is, the **DL_mib_t** structure). A process cannot use this **ioctl** to change the *ifPhyAddress*, the *ifDescrLen*, or the text of the description fields.

DLIOCGENADDR
: Returns the Ethernet address in network order.

DLIOCGLPCFLG
: Returns the state of the local packet copy flag in the *ioc_rval* of the **iocblk** structure. The local copy flag determines whether packets looped back by the driver should also be sent to the network. A non-zero value indicates that frames should also be be sent to the network after being looped back. The default value of this flag is zero.

DLIOCSLPCFLG
: Allows a privileged process to set the local packet copy flag, causing all packets looped back by the driver to be sent to the network as well.

DLIOCGPROMISC
: Returns the value of the promiscuous flag in the *ioc_rval* field of the **iocblk** structure. A non-zero value indicates that the Ethernet interface will receive all frames on the network. The default value of this flag is zero.

DLIOCSPROMISC
: Allows a privileged process to toggle the current state of the promiscuous flag. When the flag is set, the driver captures all frames from the network. Processes that are bound to a promiscuous SAP, or to a SAP that matches the type field of the received frame, receive a copy of the frame.

Installation

You select the interrupt level for the i596 at package installation time. The following interrupts are valid: 9, 10, 11, or 15.

This interrupt should match the "LAN IRQ Level" value configured by the EISA Configuration Utility (ECU). In addition, the "Onboard LAN" must be enabled and the "LAN Media Type" set to either AUI or Twisted Pair, whichever is appropriate.

You can also set the interrupt level in the **/etc/conf/sdevice.d/i596** file.

Configuration

The **i596** driver has three configurable parameters in the **/etc/conf/pack.d/i596/space.c** file. Any changes to this file must be followed by a rebuild of the kernel and a reboot of the system for the changes to take effect.

The configurable parameters are:

N_SAPS
: Defines the number of SAPs that can be bound at any one time. This value should be only slightly larger than anticipated SAP usage. A typical TCP/IP system would require two SAPs (0x800 and 0x806). A large value will degrade performance and increase memory usage.

STREAMS_LOG
: Defines whether the driver should log debugging messages to the STREAMS logger for the **strace**(1M) utility to display. The module ID used with **strace** is 2101. A value of 0 indicates that STREAMS debug messages should not be generated. A value of 1 enables STREAMS debug messages. You can make the driver temporarily log messages by changing the value of **i596strlog** (a 4-byte integer) to 1 using the kernel debugger.

  STREAMS tracing should only be performed when debugging a network problem. It can cause a severe performance degradation if you use full **i596** STREAMS logging.

IFNAME
: This parameter is important only in a TCP/IP networking environment. It defines the string used in displaying network statistics. This string should match the logical interface name assigned in **/etc/inet/strcf** file and with **ifconfig**(1M) commands used in **/etc/inet/rc.inet** configuration script.

Errors

The **i596** driver can return the following error codes:

ENXIO Invalid major number or board is not installed.

ECHRNG
: No minor devices left if configured as a cloned device. Increase **N_SAP** value in **/etc/conf/pack.d/i596/space.c** Invalid minor device number if configured as a non-cloned device.

EPERM An **ioctl** was issued without the appropriate privilege.

EINVAL
: An **ioctl** was issued that did not supply a required input and/or output buffer.

DL_NOTSUPPORTED
: The requested service primitive is not supported.

DL_BADPRIM
: An unknown service primitive was requested.

DL_OUTSTATE
: **DL_BIND_REQ** was issued when the Stream was bound, or **DL_UNBIND_REQ** or **DL_UNITDATA_REQ** was issued when the Stream was unbound.

DL_ACCESS
: An attempt was made to bind to **PROMISCUOUS_SAP** with insufficient privilege.

DL_BOUND
: The requested SAP is already bound. A privileged process may bind to an already bound SAP.

DL_NOTINIT
: **DL_UNITDATA_REQ** was issued on an Ethernet board that has gone offline due to an error.

DL_BADDATA
: DL_UNITDATA_REQ was issued with a data size that was either larger than the SPDU maximum or smaller than the SPDU minimum.

Files

/dev/i596_0
/etc/conf/pack.d/i596/space.c
/etc/conf/sdevice.d/i596

REFERENCES

open(2), getmsg(2), ioctl(2), putmsg(2)

NAME

ibmtok – IBM Token Ring Driver

SYNOPSIS

```
#include <sys/dlpi.h>
#include <sys/ibmtokhw.h>
#include <sys/ibmtok.h>

fd = open ("/dev/ibmtok_0", O_RDWR)
```

DESCRIPTION

The **ibmtok** driver provides a data link interface to both the 16/4 and the 16/4A token ring adapters from IBM. It is a STREAMS-based driver that is compatible with the Data Link Provider Interface (DLPI) and Logical Link Interface (LLI) software interfaces.

The **ibmtok** driver can operate as a cloned or non-cloned device.

A process must issue a DL_BIND_REQ primitive to receive frames from the network. The process must specify the *dl_sap* field of the **dl_bind_req_t** structure. The *type* field of an incoming frame is compared to the *dl_sap* value. If the values are equal, it is placed on the STREAMS read queue of the process.

Received frames are delivered in **dl_unitdata_ind_t** structures. The source and destination address each contain a 6-byte Ethernet address, followed by the 1- or 2-byte type value.

Any user process that has a Stream bound (using a **DL_BIND_REQ**) to the token ring driver should explicitly unbind (using **DL_UNBIND_REQ**) before closing the Stream. Any destructive close (closing without unbind) will render the SAP useless until the board can be closed and opened again.

USAGE

You can select the interrupt level, base I/O address, ROM address and the shared RAM address for the token ring card at package installation time. While interrupts 2, 3, 6, and 7 are valid for the AT version of the card, 2, 3, 10, and 12 should be used for the MCA version of the card. The shared RAM is a movable section of memory and can be 8K, 16K, 32K or 64K. The starting address of the shared RAM is a function of the size and should fall on appropriate boundaries. For example, the starting address of a 16K RAM should only be on 16K boundaries. Similarly, you can also choose the starting address of an 8K ROM. The ROM contains adapter configuration information and is also used to control the adapter.

The base I/O address of the card can be one of the following addresses:

0xA20	0xA23
0xA24	0xA27

You can also set these parameters in the **/etc/conf/sdevice.d/ibmtok** file.

Configuration

The **ibmtok** driver has four configurable parameters in the **/etc/conf/pack.d/ibmtok/space.c** file. Any changes to this file must be followed by a rebuild of the kernel and a reboot of the system for the changes to take effect.

The configurable parameters are:

`tok_nbr_rcv_buffers`
: Defines the number of receive buffers that are minimally required. The receive buffers are allocated in the shared RAM. Though the default value is 20, the adapter can and will allocate all unused memory to the receive buffers.

`tok_rcv_buff_size`
: Defines the size of the receive buffers. The default value is 264.

`tok_tx_buff_size`
: The size of the transmit buffer. The default value is 2048 bytes.

`tok_nbr_tx_buffers`
: The number of transmit buffers. The default value is 1.

Ioctl Calls

The following **`ioctls`** are supported:

DLGDEVSTAT
: Returns the **`tokstat`** structure, which contains the information about the number of packets and bytes transmitted and received for that particular SAP.

```
/*
 *  Per sap statistics structure.
 */
typedef struct {
  ulong    toks_xpkts;     /*The number of packets transmitted */
  ulong    toks_xbytes;    /*The number of bytes transmitted */
  ulong    toks_rpkts;     /*The number of packets received */
  ulong    toks_rbytes;    /*The number of bytes received */
} tokdevstat;
```

DLGADDR
: Returns the Ethernet address in network order.

Error Codes

The **`ibmtok`** driver can return the following error codes:

ENXIO Invalid major number, board not installed, or a non-functional board.

ECHRNG
: No minor devices left, if configured as a cloned device.

EINTR Signal terminating a process sleeping on an event.

EINVAL
: An **`ioctl`** was issued that did not supply a required input and/or output buffer.

`DL_NOTSUPPORTED`
: Requested service primitive is not supported.

`DL_BADPRIM`
: Unknown service primitive was requested.

`DL_OUTSTATE`
: `DL_BIND_REQ` was issued when the stream was bound, or `DL_UNBIND_REQ` or `DL_UNITDATA_REQ` was issued when the stream was unbound.

`DL_BOUND`
: The requested SAP is already bound.

`DL_NOTINIT`
: A service primitive was issued to a token ring board that has gone offline due to an error.

`DL_BADDATA`
: `DL_UNITDATA_REQ` was issued with a data size that was either larger than the SPDU maximum or smaller than the SPDU minimum.

Files

`/dev/ibmtok*`
`/etc/conf/pack.d/ibmtok/space.c`
`/etc/conf/sdevice.d/ibmtok`

REFERENCES

`getmsg`(2), `ioctl`(2), `open`(2), `putmsg`(2)

NAME

ICMP – Internet Control Message Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip_icmp.h>

s = socket(AF_INET, SOCK_RAW, proto);

t = t_open("/dev/icmp", O_RDWR);
```

DESCRIPTION

ICMP is the error and control message protocol used by the Internet protocol family. It is used by the kernel to handle and report errors in protocol processing. It may also be accessed by programs using the socket interface or the Transport Level Interface (TLI) for network monitoring and diagnostic functions. When used with the socket interface, a raw socket type is used. The protocol number for ICMP, used in the *proto* parameter to the socket call, can be obtained from **getprotobyname**() [see **getprotoent**(3N)]. ICMP file descriptors and sockets are connectionless, and are normally used with the **t_sndudata** / **t_rcvudata** and the **sendto()** / **recvfrom()** calls [see send(3N) and recv(3N)].

Outgoing packets automatically have an Internet Protocol (IP) header prepended to them. Incoming packets are provided to the user with the IP header and options intact.

ICMP is an datagram protocol layered above IP. It is used internally by the protocol code for various purposes including routing, fault isolation, and congestion control. Receipt of an ICMP redirect message will add a new entry in the routing table, or modify an existing one. ICMP messages are routinely sent by the protocol code. Received ICMP messages may be reflected back to users of higher-level protocols such as TCP or UDP as error returns from system calls. A copy of all ICMP message received by the system is provided to every holder of an open ICMP socket or TLI descriptor.

SEE ALSO

getprotoent(3N), **inet**(7), **ip**(7), **recv**(3N), **routing**(4), **send**(3N), **t_rcvudata**(3N), **t_sndudata**(3N)

Postel, Jon, *Internet Control Message Protocol — DARPA Internet Program Protocol Specification*, RFC 792, Network Information Center, SRI International, Menlo Park, Calif., September 1981

DIAGNOSTICS

A socket operation may fail with one of the following errors returned:

EISCONN	An attempt was made to establish a connection on a socket which already has one, or when trying to send a datagram with the destination address specified and the socket is already connected.
ENOTCONN	An attempt was made to send a datagram, but no destination address is specified, and the socket has not been connected.

`ENOBUFS`	The system ran out of memory for an internal data structure.
`EADDRNOTAVAIL`	An attempt was made to create a socket with a network address for which no network interface exists.

NOTES

Replies to ICMP echo messages which are source routed are not sent back using inverted source routes, but rather go back through the normal routing mechanisms.

NAME

ie6 – 3C503 3Com Ethernet Driver

SYNOPSIS

```
#include <sys/dlpi.h>
#include <sys/dlpi_ether.h>
#include <sys/ie6.h>

fd = open ("/dev/ie6_0", O_RDWR)
```

DESCRIPTION

The **ie6** driver provides a data link interface to the 3C503 Ethernet controller from 3Com. It is a STREAMS-based driver, compatible with the Data Link Provider Interface (DLPI) and Logical Link Interface (LLI) software interfaces.

The **ie6** driver supports both **DL_ETHER** and **DL_CSMACD** for MAC type, **DL_CL_ETHER** for service mode, and **DL_STYLE1** for provider style. The driver can operate as a cloned or non-cloned device.

A process must issue a **DL_BIND_REQ** primitive to receive frames from the network. This primitive includes a **dl_bind_req_t** structure.

The process must specify the *dl_sap* field of the **dl_bind_req_t** structure in host order. The type field of an incoming frame is converted to host order and compared to the *dl_sap* value. If the values are equal, the frame is placed on the STREAMS read queue of the requesting process. A privileged process may set the *dl_sap* field to **PROMISCUOUS_SAP**. The **PROMISCUOUS_SAP** matches all incoming frames.

A privileged process may bind to an SAP already bound by another process. In cases where a frame qualifies to be sent to more than one process, independent copies of the frame are made and placed on the STREAMS read queue of each process.

Received frames are delivered in **dl_unitdata_ind_t** structures. The source and destination address each contain a 6-byte Ethernet address, followed by the 2-byte type value, written in network order.

USAGE

ioctl Calls

The following **ioctl**s are supported:

DLIOCGMIB

Returns the **DL_mib_t** structure, which contains the Management Information Base (MIB). The MIB holds the Ethernet statistics kept in the driver.

```
/*
 *  Ether statistics structure.
 */
typedef struct {
  ulong_tetherAlignErrors;    /* Frame alignment errors */
  ulong_tetherCRCerrors;      /* CRC errors */
  ulong_tetherMissedPkts;     /* Packet overflow or missed inter */
  ulong_tetherOverrunErrors;  /* Overrun errors */
  ulong_tetherUnderrunErrors; /* Underrun errors */
  ulong_tetherCollisions;     /* Total collisions */
 ulong_t etherAbortErrors;    /* Transmits aborted at interface */
  ulong_tetherCarrierLost;    /* Carrier sense signal lost */
  ulong_tetherReadqFull;      /* STREAMS read queue full */
  ulong_tetherRcvResources;   /* Receive resource alloc faliure
*/
  ulong_tetherDependent1;     /* Device dependent statistic */
  ulong_tetherDependent2;     /* Device dependent statistic */
  ulong_tetherDependent3;     /* Device dependent statistic */
  ulong_tetherDependent4;     /* Device dependent statistic */
  ulong_tetherDependent5;     /* Device dependent statistic */
} DL_etherstat_t;

/*
 *  Interface statistics compatible with MIB II SNMP requirements.
 */
typedef struct {
  int     ifIndex;            /* 1 through ifNumber */
  int     ifDescrLen;         /* len of desc. following this struct */
  int     ifType;             /* type of interface */
  int     ifMtu;              /* datagram size that can be sent/rcv */
  ulong_tifSpeed;             /* estimate of bandwith in bits PS */
  uchar_tifPhyAddress[DL_MAC_ADDR_LEN];/* Ethernet Address */
  int     ifAdminStatus;      /* desired state of the interface */
  int     ifOperStatus;       /* current state of the interface */
  ulong_tifLastChange;        /* sysUpTime when state was entered */
  ulong_tifInOctets;          /* octets received on interface */
  ulong_tifInUcastPkts;       /* unicast packets delivered */
  ulong_tifInNUcastPkts;      /* non-unicast packets delivered */
  ulong_tifInDiscards;        /* good packets received but dropped */
  ulong_tifInErrors;          /* packets received with errors */
  ulong_tifInUnknownProtos;   /* packets recv'd to unbound proto
*/
  ulong_tifOutOctets;         /* octets transmitted on interface '*/
  ulong_tifOutUcastPkts;      /* unicast packets transmited */
  ulong_tifOutNUcastPkts;     /* non-unicast packets transmited */
  ulong_tifOutDiscards;       /* good outbound packets dropped */
  ulong_tifOutErrors;         /* number of transmit errors */
  ulong_tifOutQlen;           /* length of output queue */
  DL_etherstat_t ifSpecific;  /* Ethernet specific stats */
} DL_mib_t;
```

The values in the MIB are compatible with those needed by the SNMP protocol.

The *ifDescrLen* field indicates the length of the null-terminated description string that immediately follows the `DL_mib_t` structure.

There are three fields in the MIB that are specific to the **ie6** driver: The *ifSpecific.etherDependent1* field tracks the number of times the transceiver failed to transmit a collision signal after transmission of a packet. The *ifSpecific.etherDependent2* field tracks the number of collisions that occurred after a slot time (out-of-window collisions). The *ifSpecific.etherDependent3* field tracks the number of times a transmit interrupt timeout condition occurred.

DLIOCSMIG
Allows a privileged process to initialize the values in the MIB (that is, the **DL_mib_t** structure). A process cannot use this **ioctl** to change the **ifPhyAddress**, the **ifDescrLen**, or the text of the description fields.

DLIOCGENADDR
Returns the Ethernet address in network order.

DLIOCGLPCFLG
Returns the state of the local packet copy flag in the *ioc_rval* field of the **iocblk** structure. The local copy flag determines if packets looped back by the driver should also be sent to the network. A non-zero value indicates that frames should also be be sent to the network after being looped back. The default value of this flag is zero.

DLIOCSLPCFLG
Allows a privileged process to set the local packet copy flag, causing all packets looped back by the driver to also be sent to the network.

DLIOCGPROMISC
Returns the value of the promiscuous flag in the *ioc_rval* of the **iocblk** structure. A non-zero value indicates that the Ethernet interface will receive all frames on the network. The default value of this flag is zero.

DLIOCSPROMISC
Allows a privileged process to toggle the current state of the promiscuous flag. When the flag is non-zero, the driver captures all frames from the network. Processes that are bound to the promiscuous SAP, or to an SAP that matches the type field of the received frame, receive a copy of the frame.

DLIOCGETMULTI
Returns the current list of multicast addresses (if it exists).

DLIOCADDMULTI
Allows a privileged process to add a new multicast address and enable its reception. A 6-byte buffer pointing to the multicast address must be passed as the parameter.

DLIOCDELMULTI
Allows a privileged process to delete a multicast address by passing a 6-byte multicast address as the parameter.

Configuration

The **ie6** driver has four configurable parameters in the **/etc/conf/pack.d/ie6/space.c** file. If you change this file, you must rebuild the kernel and reboot the system for the changes to take effect.

The configurable parameters are:

N_SAPS
: Defines the number of SAPs that can be bound at any one time. This value should be only slightly larger than anticipated SAP usage. A typical TCP/IP system requires two SAPs (0x800 and 0x806). If you assign too large a value to this parameter, system performance and memory usage may suffer.

CABLE_TYPE
: Defines the type of Ethernet cable attached to the Ethernet controller card. A value of 0 specifies thin Ethernet cable with a BNC connector. A value of 1 specifies thick Ethernet cable with a AUI connector.

STREAMS_LOG
: Defines whether the driver should log debugging messages to the STREAMS logger for the **strace**(1M) utility to display. The module ID used with **strace** is 2101. A value of 0 indicates that no STREAMS debug messages should be generated. A value of 1 enables STREAMS debug messages to be generated. You can also direct the driver to log messages temporarily by using the kernel debugger to change the value of **ie6strlog** (a 4-byte integer) to 1.

  Use STREAMS tracing only when debugging a network problem, because system performance suffers when full **ie6** STREAMS logging is in progress.

IFNAME
: This parameter is important only in a TCP/IP networking environment. It defines the string used in displaying network statistics. This string should match the logical interface name assigned in the **/etc/confnet.d/inet/interfaces** file and with **ifconfig**(1M) commands used in the **/etc/inet/rc.inet** configuration script.

Errors

The **ie6** driver can return the following error codes:

ENXIO Invalid major number or board is not installed.

ECHRNG
: No minor devices left if configured as a cloned device. Increase **N_SAP** value in **/etc/conf/pack.d/ie6/space.c** Invalid minor device number if configured as a non-cloned device.

EPERM An **ioctl** was made without the appropriate privilege.

EINVAL
: An **ioctl** was made that did not supply a required input and/or output buffer.

DL_NOTSUPPORTED
: Requested service primitive is not supported.

DL_BADPRIM
: Unknown service primitive was requested.

DL_OUTSTATE
: **DL_BIND_REQ** was issued when the Stream was bound, or **DL_UNBIND_REQ** or **DL_UNITDATA_REQ** were issued when the Stream was unbound.

DL_ACCESS
: An attempt was made to bind to **PROMISCUOUS_SAP** with insufficient privilege.

DL_BOUND
: The requested SAP is already bound. A privileged process may bind to an already bound SAP.

DL_NOTINIT
: **DL_UNITDATA_REQ** was issued on an Ethernet board that has gone offline due to an error.

DL_BADDATA
: **DL_UNITDATA_REQ** was issued with a data size that was either larger than the SPDU maximum or smaller than the SPDU minimum.

Files

`/dev/ie6_*`
`/etc/conf/pack.d/ie6/space.c`

REFERENCES

getmsg(2), **ifconfig**(1M), **ioctl**(2), **open**(2), **putmsg**(2), **strace**(1M)

NAME

`if` – general properties of Internet Protocol network interfaces

DESCRIPTION

A network interface is a device for sending and receiving packets on a network. A network interface is usually a hardware device, although certain interfaces such as the loopback interface, `lo`(7), are implemented in software. Network interfaces used by the Internet Protocol (IP) must be STREAMS devices conforming to the Datalink Provider Interface (DLPI).

An interface becomes available to IP when it is linked below the IP STREAMS device with the `I_LINK` `ioctl()` call. This may be initiated by the kernel at boot time or by a user program some time after the system is running. Each IP interface must have a name assigned to it with the **`SIOCSIFNAME`** `ioctl()`. This name is used as a unique handle on the interface by all of the other network interface `ioctl()` calls. Each interface must be assigned an IP address with the **`SIOCSIFADDR`** `ioctl()` before it can be used. On interfaces where the network-to-link layer address mapping is static, only the network number is taken from the `ioctl()` request; the remainder is found in a hardware specific manner. On interfaces which provide dynamic network-to-link layer address mapping facilities [for example, 10Mb/s Ethernets using **`arp`**(7)], the entire address specified in the `ioctl()` is used. A routing table entry for destinations on the network of the interface is installed automatically when an interface's address is set.

IOCTLS

The following `ioctl()` calls may be used to manipulate IP network interfaces. Unless specified otherwise, the request takes an `ifreq` structure as its parameter. This structure has the form:

```
/* Interface request structure used for socket ioctl's.  All */
/* interface ioctl's must have parameter definitions which */
/* begin with ifr_name.  The remainder may be interface specific. */

struct  ifreq {
#define IFNAMSIZ     16
    char         ifr_name[IFNAMSIZ];      /* if name, for example "emd1" */
    union {
         struct  sockaddr ifru_addr;
         struct  sockaddr ifru_dstaddr;
         char    ifru_oname[IFNAMSIZ];        /* other if name */
         struct  sockaddr ifru_broadaddr;
         short   ifru_flags;
         int     ifru_metric;
         char    ifru_data[1];                /* interface dependent data */
         char    ifru_enaddr[6];
    } ifr_ifru;
#define ifr_addr      ifr_ifru.ifru_addr      /* address */
#define ifr_dstaddr   ifr_ifru.ifru_dstaddr   /* other end of p-to-p link */
#define ifr_oname     ifr_ifru.ifru_oname     /* other if name */
#define ifr_broadaddr ifr_ifru.ifru_broadaddr /* broadcast address */
#define ifr_flags     ifr_ifru.ifru_flags     /* flags */
#define ifr_metric    ifr_ifru.ifru_metric    /* metric */
#define ifr_data      ifr_ifru.ifru_data      /* for use by interface */
#define ifr_enaddr    ifr_ifru.ifru_enaddr    /* ethernet address */
};
```

SIOCSIFADDR	Set interface address. Following the address assignment, the initialization routine for the interface is called.
SIOCGIFADDR	Get interface address.
SIOCSIFDSTADDR	Set point to point address for interface.
SIOCGIFDSTADDR	Get point to point address for interface.
SIOCSIFFLAGS	Set interface flags field. If the interface is marked down, any processes currently routing packets through the interface are notified. The interface can be marked up or down by using **ifconfig**(1M).
SIOCGIFFLAGS	Get interface flags.
SIOCGIFCONF	Get interface configuration list. This request takes an **ifconf** structure (see below) as a value-result parameter. The **ifc_len** field should be initially set to the size of the buffer pointed to by **ifc_buf**. On return it will contain the length, in bytes, of the configuration list.

The **ifconf** structure has the form:

```
/*
 * Structure used in SIOCGIFCONF request.
 * Used to retrieve interface configuration
 * for machine (useful for programs which
 * must know all networks accessible).
 */
structifconf {
      int     ifc_len;           /* size of associated buffer */
      union {
              caddr_t ifcu_buf;
              struct  ifreq *ifcu_req;
      } ifc_ifcu;
#define ifc_buf ifc_ifcu.ifcu_buf  /* buffer address */
#define ifc_req ifc_ifcu.ifcu_req  /* array of structures returned */
};
```

SIOCSIFNAME
Set the name of the interface.

SEE ALSO

arp(7), **ip**(7), **ifconfig**(1M), **lo**(7)

NAME

`imx586` – IMXLAN586 Intel Ethernet Driver

SYNOPSIS

```
#include <sys/dlpi.h>
#include <sys/dlpi_ether.h>
#include <sys/imx586.h>

fd = open ("/dev/imx586_0", O_RDWR)
```

DESCRIPTION

The **imx586** driver provides a data link interface to the iMX-LAN/586 Ethernet controller from Intel. It is a STREAMS-based driver, compatible with the Data Link Provider Interface (DLPI) and Logical Link Interface (LLI) software interfaces.

The **imx586** driver supports both **DL_ETHER** and **DL_CSMACD** for MAC type, **DL_CL_ETHER** for service mode, and **DL_STYLE1** for provider style. The driver can operate as a cloned or non-cloned device.

A process must issue a **DL_BIND_REQ** primitive to receive frames from the network. This primitive includes a **dl_bind_req_t** structure.

The process must specify the *dl_sap* field of the **dl_bind_req_t** structure in host order. The type field of an incoming frame is converted to host order and compared to the *dl_sap* value. If the values are equal, the frame is placed on the STREAMS read queue of the requesting process. A privileged process may set the *dl_sap* field to **PROMISCUOUS_SAP**. The **PROMISCUOUS_SAP** matches all incoming frames.

A privileged process may bind to an SAP already bound by another process. In cases where a frame qualifies to be sent to more than one process, independent copies of the frame are made and placed on the STREAMS read queue of each process.

Received frames are delivered in **dl_unitdata_ind_t** structures. The source and destination address each contain a 6-byte Ethernet address, followed by a 2-byte type value, written in network order.

USAGE

ioctl Calls

The following `ioctl`s are supported:

DLIOCGMIB

Returns the **DL_mib_t** structure, which contains the Management Information Base (MIB). The MIB holds the Ethernet statistics kept in the driver.

```
/*
 *  Ether statistics structure.
 */
typedef struct {
  ulong_tetherAlignErrors;    /* Frame alignment errors */
  ulong_tetherCRCerrors;      /* CRC errors */
  ulong_tetherMissedPkts;     /* Packet overflow or missed inter */
  ulong_tetherOverrunErrors;  /* Overrun errors */
  ulong_tetherUnderrunErrors; /* Underrun errors */
  ulong_tetherCollisions;     /* Total collisions */
  ulong_tetherAbortErrors;    /* Transmits aborted at interface */
  ulong_tetherCarrierLost;    /* Carrier sense signal lost */
  ulong_tetherReadqFull;      /* STREAMS read queue full */
  ulong_tetherRcvResources;   /* Receive resource alloc faliure
*/
  ulong_tetherDependent1;     /* Device dependent statistic */
  ulong_tetherDependent2;     /* Device dependent statistic */
  ulong_tetherDependent3;     /* Device dependent statistic */
  ulong_tetherDependent4;     /* Device dependent statistic */
  ulong_tetherDependent5;     /* Device dependent statistic */
} DL_etherstat_t;

/*
 *  Interface statistics compatible with MIB II SNMP requirements.
 */
typedef struct {
  int     ifIndex;            /* 1 through ifNumber */
  int     ifDescrLen;         /* len of desc. following this struct */
  int     ifType;             /* type of interface */
  int     ifMtu;              /* datagram size that can be sent/rcv */
  ulong_tifSpeed;             /* estimate of bandwith in bits PS */
  uchar_tifPhyAddress[DL_MAC_ADDR_LEN];/* Ethernet Address */
  int     ifAdminStatus;      /* desired state of the interface */
  int     ifOperStatus;       /* current state of the interface */
  ulong_tifLastChange;        /* sysUpTime when state was entered */
  ulong_tifInOctets;          /* octets received on interface */
  ulong_tifInUcastPkts;       /* unicast packets delivered */
  ulong_tifInNUcastPkts;      /* non-unicast packets delivered */
  ulong_tifInDiscards;        /* good packets received but dropped */
  ulong_tifInErrors;          /* packets received with errors */
  ulong_tifInUnknownProtos;   /* packets recv'd to unbound proto
*/
  ulong_tifOutOctets;         /* octets transmitted on interface '*/
  ulong_tifOutUcastPkts;      /* unicast packets transmited */
  ulong_tifOutNUcastPkts;     /* non-unicast packets transmited */
  ulong_tifOutDiscards;       /* good outbound packets dropped */
  ulong_tifOutErrors;         /* number of transmit errors */
  ulong_tifOutQlen;           /* length of output queue */
  DL_etherstat_t ifSpecific;  /* Ethernet specific stats */
} DL_mib_t;
```

The values in the MIB are compatible with those needed by the SNMP protocol.

The *ifDescrLen* field indicates the length of the null-terminated description string that immediately follows the **DL_mib_t** structure.

There are three fields in the MIB that are specific to the **imx586** driver: The *ifSpecific.etherDependent1* field tracks the number of times the transceiver failed to transmit a collision signal after transmission of a packet. The *ifSpecific.etherDependent2* field tracks the number of collisions that occurred after a slot time (out-of-window collisions). The *ifSpecific.etherDependent3* field tracks the number of times a transmit interrupt timeout condition occurred.

DLIOCSMIG
Allows a privileged process to initialize the values in the MIB (that is, the **DL_mib_t** structure). A process cannot use this **ioctl** to change the **ifPhyAddress**, the **ifDescrLen**, or the text of the description fields.

DLIOCGENADDR
Returns the Ethernet address in network order.

DLIOCGLPCFLG
Returns the state of the local packet copy flag in the *ioc_rval* field of the **iocblk** structure. The local copy flag determines if packets looped back by the driver should also be sent to the network. A non-zero value indicates that frames should also be be sent to the network after being looped back. The default value of this flag is zero.

DLIOCSLPCFLG
Allows a privileged process to set the local packet copy flag, causing all packets looped back by the driver to also be sent to the network.

DLIOCGPROMISC
Returns the value of the promiscuous flag in the *ioc_rval* of the **iocblk** structure. A non-zero value indicates that the Ethernet interface will receive all frames on the network. The default value of this flag is zero.

DLIOCSPROMISC
Allows a privileged process to toggle the current state of the promiscuous flag. When the flag is non-zero, the driver captures all frames from the network. Processes that are bound to the promiscuous SAP, or to an SAP that matches the type field of the received frame, receive a copy of the frame.

DLIOCGETMULTI
Returns the current list of multicast addresses (if it exists).

DLIOCADDMULTI
Allows a privileged process to add a new multicast address and enable its reception. A 6-byte buffer pointing to the multicast address must be passed as the parameter.

DLIOCDELMULTI
Allows a privileged process to delete a multicast address by passing a 6-byte multicast address as the parameter.

Configuration

The **imx586** driver has four configurable parameters in the **/etc/conf/pack.d/imx586/space.c** file. If you change this file, you must rebuild the kernel and reboot the system for the changes to take effect.

The configurable parameters are:

N_SAPS
: Defines the number of SAPs that can be bound at any one time. This value should be only slightly larger than anticipated SAP usage. A typical TCP/IP system requires two SAPs (0x800 and 0x806). If you assign too large a value to this parameter, system performance and memory usage may suffer.

STREAMS_LOG
: Defines whether the driver should log debugging messages to the STREAMS logger for the **strace**(1M) utility to display. The module ID used with **strace** is 2101. A value of 0 indicates that no STREAMS debug messages should be generated. A value of 1 enables STREAMS debug messages to be generated. You can also direct the driver to log messages temporarily by using the kernel debugger to change the value of **imx586strlog** (a 4-byte integer) to 1.

  Use STREAMS tracing only when debugging a network problem, because system performance suffers when full **imx586** STREAMS logging is in progress.

IFNAME
: This parameter is important only in a TCP/IP networking environment. It defines the string used in displaying network statistics. This string should match the logical interface name assigned in the **/etc/confnet.d/inet/interfaces** file and with **ifconfig**(1M) commands used in the **/etc/inet/rc.inet** configuration script.

Errors

The **imx586** driver can return the following error codes:

ENXIO Invalid major number or board is not installed.

ECHRNG
: No minor devices left if configured as a cloned device. Increase **N_SAP** value in **/etc/conf/pack.d/imx586/space.c** Invalid minor device number if configured as a non-cloned device.

EPERM An **ioctl** was made without the appropriate privilege.

EINVAL
: An **ioctl** was made that did not supply a required input and/or output buffer.

DL_NOTSUPPORTED
: Requested service primitive is not supported.

DL_BADPRIM
: Unknown service primitive was requested.

DL_OUTSTATE
: **DL_BIND_REQ** was issued when the Stream was bound, or **DL_UNBIND_REQ** or **DL_UNITDATA_REQ** were issued when the Stream was unbound.

`DL_ACCESS`
: An attempt was made to bind to **`PROMISCUOUS_SAP`** with insufficient privilege.

`DL_BOUND`
: The requested SAP is already bound. A privileged process may bind to an already bound SAP.

`DL_NOTINIT`
: **`DL_UNITDATA_REQ`** was issued on an Ethernet board that has gone offline due to an error.

`DL_BADDATA`
: **`DL_UNITDATA_REQ`** was issued with a data size that was either larger than the SPDU maximum or smaller than the SPDU minimum.

Files

`/dev/imx586_*`
`/etc/conf/pack.d/imx586/space.c`

REFERENCES

`getmsg`(2), **`ifconfig`**(1M), **`ioctl`**(2), **`open`**(2), **`putmsg`**(2), **`strace`**(1M)

NAME

inet – Internet protocol family

SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>
```

DESCRIPTION

The Internet protocol family implements a collection of protocols which are centered around the *Internet Protocol* (IP) and which share a common address format. The Internet family protocols can be accessed via the socket interface, where they support the **SOCK_STREAM**, **SOCK_DGRAM**, and **SOCK_RAW** socket types, or the Transport Level Interface (TLI), where they support the connectionless (**T_CLTS**) and connection oriented (**T_COTS_ORD**) service types.

PROTOCOLS

The Internet protocol family comprises the Internet Protocol (IP), the Address Resolution Protocol (ARP), the Internet Control Message Protocol (ICMP), the Transmission Control Protocol (TCP), and the User Datagram Protocol (UDP).

TCP supports the socket interface's **SOCK_STREAM** abstraction and TLI's **T_COTS_ORD** service type. UDP supports the **SOCK_DGRAM** socket abstraction and the TLI **T_CLTS** service type. See **tcp**(7) and **udp**(7). A direct interface to IP is available via both TLI and the socket interface; See **ip**(7). ICMP is used by the kernel to handle and report errors in protocol processing. It is also accessible to user programs; see **icmp**(7). ARP is used to translate 32-bit IP addresses into 48-bit Ethernet addresses; see **arp**(7).

The 32-bit IP address is divided into network number and host number parts. It is frequency-encoded; The most-significant bit is zero in Class A addresses, in which the high-order 8 bits represent the network number. Class B addresses have their high order two bits set to 10 and use the high-order 16 bits as the network number field. Class C addresses have a 24-bit network number part of which the high order three bits are 110. Rather than using the default class A, B, or C address, a subnet mask can be used for the network interface. See "Setting Up Subnets" in the "Expanding Your TCP/IP Network" chapter in *Network Administration* for more information on subnet masks. Subnet addressing is enabled and examined by the following **ioctl**(2) commands; They have the same form as the **SIOCSIFADDR** command [see **if**(7)].

SIOCSIFNETMASK	Set interface network mask. The network mask defines the network part of the address; If it contains more of the address than the address type would indicate, then subnets are in use.
SIOCGIFNETMASK	Get interface network mask.

ADDRESSING

IP addresses are four byte quantities, stored in network byte order. IP addresses should be manipulated using the byte order conversion routines [see **byteorder**(3N)].

Addresses in the Internet protocol family use the following structure:

```
struct sockaddr_in {
        short    sin_family;
        u_short  sin_port;
        struct   in_addr sin_addr;
        char     sin_zero[8];
};
```

Library routines are provided to manipulate structures of this form; See **inet**(3N).

The **sin_addr** field of the **sockaddr_in** structure specifies a local or remote IP address. Each network interface has its own unique IP address. The special value **INADDR_ANY** may be used in this field to effect wildcard matching. Given in a **bind**(3N) call, this value leaves the local IP address of the socket unspecified, so that the socket will receive connections or messages directed at any of the valid IP addresses of the system. This can prove useful when a process neither knows nor cares what the local IP address is or when a process wishes to receive requests using all of its network interfaces. The **sockaddr_in** structure given in the **bind**(3N) call must specify an **in_addr** value of either **IPADDR_ANY** or one of the system's valid IP addresses. Requests to bind any other address will elicit the error **EADDRNOTAVAI**. When a **connect**(3N) call is made for a socket that has a wildcard local address, the system sets the **sin_addr** field of the socket to the IP address of the network interface that the packets for that connection are routed via.

The **sin_port** field of the **sockaddr_in** structure specifies a port number used by TCP or UDP. The local port address specified in a **bind**(3N) call is restricted to be greater than **IPPORT_RESERVED** (defined in **<netinet/in.h>**) unless the creating process is running as the super-user, providing a space of protected port numbers. In addition, the local port address must not be in use by any socket of same address family and type. Requests to bind sockets to port numbers being used by other sockets return the error **EADDRINUSE**. If the local port address is specified as 0, then the system picks a unique port address greater than **IPPORT_RESERVED**. A unique local port address is also picked when a socket which is not bound is used in a **connect**(3N) or **sendto**() [see **send**(3N)] call. This allows programs which do not care which local port number is used to set up TCP connections by simply calling **socket**(3N) and then **connect**(3N), and to send UDP datagrams with a **socket**(3N) call followed by a **sendto**() [see **send**(3N)] call.

Although this implementation restricts sockets to unique local port numbers, TCP allows multiple simultaneous connections involving the same local port number so long as the remote IP addresses or port numbers are different for each connection. Programs may explicitly override the socket restriction by setting the **SO_REUSEADDR** socket option with **setsockopt** [see **getsockopt**(3N)].

TLI applies somewhat different semantics to the binding of local port numbers. These semantics apply when Internet family protocols are used via the TLI.

SEE ALSO

arp(7), **bind**(3N), **byteorder**(3N), **connect**(3N), **icmp**(7), **if**(7), **ioctl**(2), **ip**(7), **gethostent**(3N), **getnetent**(3N), **getprotoent**(3N), **getservent**(3N), **getsockopt**(3N), **send**(3N), **socket**(3N), **tcp**(7), **udp**(7)

Network Information Center, *DDN Protocol Handbook* (3 vols.), Network Information Center, SRI International, Menlo Park, Calif., 1985

NOTES

The Internet protocol support is subject to change as the Internet protocols develop. Users should not depend on details of the current implementation, but rather the services exported.

NAME

IP – Internet Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_RAW, proto);

t = t_open ("/dev/rawip", O_RDWR);

d = open ("/dev/ip", O_RDWR);
```

DESCRIPTION

IP is the internetwork datagram delivery protocol that is central to the Internet protocol family. Programs may use IP through higher-level protocols such as the Transmission Control Protocol (TCP) or the User Datagram Protocol (UDP), or may interface directly to IP. See **tcp**(7) and **udp**(7). Direct access may be via the socket interface (using a raw socket) or the Transport Level Interface (TLI). The protocol options defined in the IP specification may be set in outgoing datagrams.

The STREAMS driver **/dev/rawip** is the TLI transport provider that provides raw access to IP. The device **/dev/ip** is the multiplexing STREAMS driver that implements the protocol processing of IP. The latter connects below to datalink providers [interface drivers, see **if**(7)], and above to transport providers such as TCP and UDP.

Raw IP sockets are connectionless and are normally used with the **sendto( )** and **recvfrom( )** calls, [see **send**(3N) and **recv**(3N)] although the **connect**(3N) call may also be used to fix the destination for future datagrams [in which case the **read**(2) or **recv**(3N) and **write**(2) or **send**(3N) calls may be used]. If **proto** is zero, the default protocol, **IPPROTO_RAW**, is used. If **proto** is non-zero, that protocol number will be set in outgoing datagrams and will be used to filter incoming datagrams. An IP header will be generated and prepended to each outgoing datagram; received datagrams are returned with the IP header and options intact.

A single socket option, **IP_OPTIONS**, is supported at the IP level. This socket option may be used to set IP options to be included in each outgoing datagram. IP options to be sent are set with **setsockopt( )** [see **getsockopt**(3N)]. The **getsockopt**(3N) call returns the IP options set in the last **setsockopt( )** call. IP options on received datagrams are visible to user programs only using raw IP sockets. The format of IP options given in **setsockopt( )** matches those defined in the IP specification with one exception: the list of addresses for the source routing options must include the first-hop gateway at the beginning of the list of gateways. The first-hop gateway address will be extracted from the option list and the size adjusted accordingly before use. IP options may be used with any socket type in the Internet family.

At the socket level, the socket option **SO_DONTROUTE** may be applied. This option forces datagrams being sent to bypass the routing step in output. Normally, IP selects a network interface to send the datagram, and possibly an intermediate gateway, based on an entry in the routing table. See **routing**(4). When **SO_DONTROUTE** is set, the datagram will be sent using the interface whose network number or full IP address matches the destination address. If no interface matches, the error **ENETUNRCH** will be returned.

Raw IP datagrams can also be sent and received using the TLI connectionless primitives.

Datagrams flow through the IP layer in two directions: from the network *up* to user processes and from user processes *down* to the network. Using this orientation, IP is layered *above* the network interface drivers and *below* the transport protocols such as UDP and TCP. The Internet Control Message Protocol (ICMP) is logically a part of IP. See **icmp**(7).

IP provides for a checksum of the header part, but not the data part of the datagram. The checksum value is computed and set in the process of sending datagrams and checked when receiving datagrams. IP header checksumming may be disabled for debugging purposes by patching the kernel variable **ipcksum** to have the value zero.

IP options in received datagrams are processed in the IP layer according to the protocol specification. Currently recognized IP options include: security, loose source and record route (LSRR), strict source and record route (SSRR), record route, stream identifier, and internet timestamp.

The IP layer will normally forward received datagrams that are not addressed to it. Forwarding is under the control of the kernel variable *ipforwarding*: if *ipforwarding* is zero, IP datagrams will not be forwarded; if *ipforwarding* is one, IP datagrams will be forwarded. *ipforwarding* is usually set to one only in machines with more than one network interface (internetwork routers). This kernel variable can be patched to enable or disable forwarding.

The IP layer will send an ICMP message back to the source host in many cases when it receives a datagram that can not be handled. A time exceeded ICMP message will be sent if the time to live field in the IP header drops to zero in the process of forwarding a datagram. A destination unreachable message will be sent if a datagram can not be forwarded because there is no route to the final destination, or if it can not be fragmented. If the datagram is addressed to the local host but is destined for a protocol that is not supported or a port that is not in use, a destination unreachable message will also be sent. The IP layer may send an ICMP source quench message if it is receiving datagrams too quickly. ICMP messages are only sent for the first fragment of a fragmented datagram and are never returned in response to errors in other ICMP messages.

The IP layer supports fragmentation and reassembly. Datagrams are fragmented on output if the datagram is larger than the maximum transmission unit (MTU) of the network interface. Fragments of received datagrams are dropped from the reassembly queues if the complete datagram is not reconstructed within a short time period.

Errors in sending discovered at the network interface driver layer are passed by IP back up to the user process.

SEE ALSO

connect(3N), **getsockopt**(3N), **icmp**(7), **if**(7), **inet**(7), **read**(2), **recv**(3N), **routing**(4), **send**(3N), **tcp**(7), **udp**(7), **write**(2)

Postel, Jon, *Internet Protocol - DARPA Internet Program Protocol Specification*, RFC 791, Network Information Center, SRI International, Menlo Park, Calif., September 1981

DIAGNOSTICS

A socket operation may fail with one of the following errors returned:

`EACCES`	A IP broadcast destination address was specified and the caller was not the privileged user.
`EISCONN`	An attempt was made to establish a connection on a socket which already had one, or to send a datagram with the destination address specified and the socket was already connected.
`EMSGSIZE`	An attempt was made to send a datagram that was too large for an interface, but was not allowed to be fragmented (such as broadcasts).
`ENETUNREACH`	An attempt was made to establish a connection or send a datagram, where there was no matching entry in the routing table, or if an ICMP destination unreachable message was received.
`ENOTCONN`	A datagram was sent, but no destination address was specified, and the socket had not been connected.
`ENOBUFS`	The system ran out of memory for fragmentation buffers or other internal data structures.
`EADDRNOTAVAIL`	An attempt was made to create a socket with a local address that did not match any network interface, or an IP broadcast destination address was specified and the network interface does not support broadcast.

The following errors may occur when setting or getting IP options:

`EINVAL`	An unknown socket option name was given.
`EINVAL`	The IP option field was improperly formed; an option field was shorter than the minimum value or longer than the option buffer provided.

NOTES

Raw sockets should receive ICMP error packets relating to the protocol; currently such packets are simply discarded.

Users of higher-level protocols such as TCP and UDP should be able to see received IP options.

NAME

kbd – generalized string translation module

DESCRIPTION

The STREAMS module **kbd** is a programmable string translation module. It performs two types of operations on an input stream: the first type is simple byte-swapping via a lookup table, the second is string translation. It is useful for code set conversion and compose-key or dead-key character production on terminals and production of overstriking sequences on printers. It also can be used for minor types of key-rebinding, expansion of abbreviations, and keyboard re-arrangement (an example of the latter would be swapping the positions of the **Y** and **Z** keys, required for German keyboards, or providing Dvorak keyboard emulation for QWERTY keyboards). The **kbdcomp**(1M) manual page discusses table construction, the input language, and contains sample uses. It is intended mainly to aid administrators in configuring the module on a particular system; the user interface to the module is only through the commands **kbdload**(1M) and **kbdset**(1).

The **kbd** module works by changing an input stream according to instructions embodied in tables. It has no built in default tables. Some tables may be loaded when the system is first brought up by pushing the module and loading standard or often-used tables [see **kbdload**(1M)] that are retained in main-memory across invocations and made available to all users. These are called public tables. Users may also load private tables at any time; these tables do not remain resident.

With the **kbdset** command, users may query the module for a list of available and attached tables, attach various tables, and set the optional per-user *hotkey*, hot-key mode, and verbose *string* for their particular invocation.

When a user attaches more than one table, the user's hot-key may be used to cycle to the next table in the list. If only one table is specified, the hot-key may be used to toggle translation on and off. When multiple tables are in use, the hot-key may be used to cycle through the list of tables. [See **kbdset**(1) for a description of the available modes.]

In its initial state, **kbd** scans input for occurrences of bytes beginning a translation sequence. When receiving such a byte, **kbd** attempts to match subsequent bytes of the input to programmed sequences. Input is buffered beginning with the byte that caused the state change and is released if a match is not found. When a match fails, the first byte of the invalid sequence is sent upstream, the buffered input is shifted, and the scan begins again with the resulting input sequence. If the current table contains an error entry, its value (one or more bytes) is substituted for the offending input byte. When a sequence is found to be valid, the entire sequence is replaced with the result string specified for it.

The **kbd** module may be used in either the read or write directions, or both simultaneously. Maps and hot-keys may be specified independently for input and output.

The **kbd** also supports the use of external kernel-resident functions as if they were tables; once declared and attached (via **kbdload** and **kbdset** respectively) they may be used as simple tables or members of composites. To accomplish this, **kbd** understands the registration functions of the **alp** module and can access any function registered with that module. Further information on external functions and their definition is contained in **alp**(7). External functions are especially useful in supporting multibyte code set conversions that would be difficult or impossible with normal **kbd** tables.

Limitations

It is not an error to attach multiple tables without defining a hot-key, but the tables will not all be accessible. It is recommended that the user's hot-key be set before loading and attaching tables to avoid unpleasant side effects when an unfamiliar arrangement is first loaded.

Each user has a limitation on the amount of memory that may be used for private and attached tables. This "quota" is controlled by the **kbd_umem** variable described below. When a user that is not a privileged user attempts to load a table or create a composite table, the quota is checked, and the load will fail if it would cause the quota to be exceeded. When a composite table is attached, the space for attachment (which requires more space than the composite table itself) is charged against this quota (attachment of simple tables is not charged against the quota). The quota is enforced only when loading new tables. Detaching temporarily from unneeded composite tables may reduce the current allocation enough to load a table that would otherwise fail because of quota enforcement. To minimize chances of failure while loading tables, it is advisable to load all required tables and make all required composite tables before attaching any of them.

Configuration Parameters

The master (or **space.c**) file contains configurable parameters.

NKBDU is the maximum number of tables that may be attached by a single user. The number should be large enough to cover uncommon cases, and must be at least 2. Default is 6.

ZUMEM, from which the variable **kbd_umem** is assigned, is the maximum number of bytes that a user (other than a privileged user) may have allocated to private tables (that is, the quota). Default is 4096.

KBDTIME is the default timer value for timeout mode. It is the number of clock ticks allowed before timing out. The value of one clock tick depends on the hardware, but is usually 1/100 or 1/60 of a second. A timeout value of 20 is 1/5 second at 100Hz; with a 60Hz clock, a value of 12 produces a 1/5 second timeout. Values from 5 to 400 inclusive are allowed by the module; if the value set for **KBDTIME** is outside this range, the module forces it to the nearest limit. (This value is only a default; users may change their particular stream to use a different value depending on their own preferences, terminal baud-rate, and typing speed.)

FILES

/usr/lib/kbd	– directory containing system standard table files.
/usr/lib/kbd/*.map	– source for some system table files.

SEE ALSO

alp(7), **kbdcomp**(1M), **kbdload**(1M), **kbdset**(1)

NOTES

NULL characters may not be used in result or input strings, because they are used as string delimiters.

NAME

keyboard – system console keyboard

DESCRIPTION

The system console has two separate parts: the keyboard and the display [see **display** (7)].

The keyboard is used to type data, and send certain control signals to the computer. UNIX system software performs terminal emulation on the console screen and keyboard, and, in doing so, makes use of several particular keys and key combinations. These keys and key combinations have special names that are unique to the UNIX system, and may or may not correspond to the keytop labels on your keyboard.

When you press a key, one of the following happens:

- An ASCII value is entered
- The meaning of another key, or keys, is changed.
- A string is sent to the computer.
- A function is initiated.

When a key is pressed (a keystroke), the keyboard sends a scancode to the computer. This scancode is interpreted by the keyboard driver. The actual code sequence delivered to the terminal input routine [see **termio** (7)] is defined by a set of internal tables in the driver. These tables can be modified by software (see the discussion of **ioctl** calls below). In addition, the driver can be instructed not to do translations, delivering the keyboard up and down scan codes directly.

Changing Meanings

The action performed by a key can be changed by using certain keys in combination. For example, the **SHIFT** key changes the ASCII values of the alphanumeric keys. Holding down the **CTRL** key while pressing another key sends a control code (such as **CTRL-d**, **CTRL-s**, and **CTRL-q**). Holding down the ALT key also modifies a key's value. The **SHIFT**, **CTRL**, and **ALT** keys can be used in combination.

Switching Screens

To change screens (virtual terminals), first run the **vtlmgr** command [see **vtlmgr**(1)]. Switch the current screen by typing **ALT-SYSREQ** (also labeled **ALT-PRINTSCRN** on some systems) followed by a key that identifies the desired screen. Any active screen can be selected by following **ALT-SYSREQ** with *Fn*, where *Fn* is one of the function keys. **F1** refers to the first virtual terminal screen, **F2** refers to the second virtual terminal screen, and so on. **ALT-SYSREQ h** (home) refers to the main console display (**/dev/console**). The next active screen can be selected with **ALT-SYSREQ n**, and the previous screen can be selected with **ALT-SYSREQ** p.

The default screen switch enable sequence (**ALT-SYSREQ**) is configurable. The **SYSREQ** table entry can be changed by software (see discussion of **ioctl** calls below).

Special Keys

The following table shows which keys on a typical console correspond to UNIX system keys. In this table, a hyphen (-) between keys means you must hold down the first key while pressing the second. The mapping between characters that generate signals and the signal generated is set with **stty**(1), and may be changed [see **stty**(1)].

Name	Keytop	Action
INTR	**DEL**	Stops current action and returns to the shell. This key is also called the **RUB OUT** or **INTERRUPT** key.
BACKSPACE	←	Deletes the first character to the left of the cursor. Note that the "cursor left" key also has a left arrow (←) on its keytop, but you cannot backspace using that key.
CTRL-d	**CTRL-D**	Signals the end of input from the keyboard; also exits current shell.
CTRL-h	**CTRL-H**	Deletes the first character to the left of the cursor. Also called the **ERASE** key.
CTRL-q	**CTRL-Q**	Restarts printing after it has been stopped with **CTRL-s**.
CTRL-s	**CTRL-S**	Suspends printing on the screen (does not stop the program).
CTRL-u	**CTRL-U**	Deletes all characters on the current line. Also called the **KILL** key.
CTRL-\	**CTRL-\**	Quits current command and creates a **core** file, if allowed. (Recommended for debugging only.)
ESCAPE	**ESC**	Special code for some programs. For example, changes from insert mode to command mode in the **vi**(1) text editor.
RETURN	(down-left arrow or **ENTER**)	Terminates a command line and initiates an action from the shell.
F*n*	**F***n*	Function key *n*. **F1-F12** are unshifted, **F13-F24** are shifted **F1-F12**, **F25-F36** are **CTRL-F1** through **F12**, and **F37-F48** are **CTRL-SHIFT-F1** through **F12**. The next **F***n* keys (**F49-F60**) are on the number pad (unshifted): **F49** - '7' **F55** - '6' **F50** - '8' **F56** - '+' **F51** - '9' **F57** - '1' **F52** - '-' **F58** - '2' **F53** - '4' **F59** - '3' **F54** - '5' **F60** - '0'

Keyboard Map

The keyboard mapping structure is defined in **/usr/include/sys/kd.h**. Each key can have ten states. The first eight states are:

– **BASE**	– **CTRL-SHIFT**
– **SHIFT**	– **ALT-SHIFT**
– **CTRL**	– **ALT-CTRL**
– **ALT**	– **ALT-CTRL-SHIFT**

The two remaining states are indicated by two special bytes. The first byte is a special state byte whose bits indicate whether the key is special in one or more of the first eight states. The second byte is one of four codes represented by the characters C, N, B, or O which indicate how the lock keys affect the particular key.

The following table describes the default keyboard mapping. All values, except for special keywords (which are described later), are ASCII character values.

Heading	Description
SCAN CODE	This column contains the scan code generated by the keyboard hardware when a key is pressed. There are no table entries for the scan code generated by releasing a key.
BASE	This column contains the normal value of a key press.
SHIFT	This column contains the value of a key press when the **SHIFT** is also being held down.
LOCK	This column indicates which lock keys affect that particular key: – **C** indicates **CAPSLOCK** – **N** indicates **NUMLOCK** – **B** indicates both – **O** indicates locking is off

The remaining columns are the values of key presses when combinations of the **CTRL**, **ALT** and **SHIFT** keys are also held down.

The **SRQTAB** column entry is included in this table to provide a simple index of the default virtual terminal key selectors to the scan code to which it is assigned. The actual **SRQTAB** table is a stand-alone table which can be read or written using the **KDGKBENT** and **KDSKBENT** `ioctl` calls.

SCAN CODE	BASE	SHIFT	CTRL	CTRL SHIFT	ALT	ALT SHIFT	ALT CTRL	ALT CTRL SHIFT	LOCK	SRQTAB
1	esc	esc	esc	esc	esc	esc	esc	esc	O	nop
2	'1'	'!'	'1'	'1'	escn	escn	nop	nop	O	nop
3	'2'	'@'	'2'	nul	escn	escn	nop	nop	O	nop
4	'3'	'#'	'3'	'3'	escn	escn	nop	nop	O	nop
5	'4'	'$'	'4'	'4'	escn	escn	nop	nop	O	nop
6	'5'	'%'	'5'	'5'	escn	escn	nop	nop	O	nop
7	'6'	'^'	'6'	rs	escn	escn	nop	nop	O	nop
8	'7'	'&'	'7'	'7'	escn	escn	nop	nop	O	nop
9	'8'	'*'	'8'	'8'	escn	escn	nop	nop	O	nop
10	'9'	'('	'9'	'9'	escn	escn	nop	nop	O	nop
11	'0'	')'	'0'	'0'	escn	escn	nop	nop	O	nop
12	'-'	'_'	'-'	ns	escn	escn	nop	nop	O	nop
13	'='	'+'	'='	'='	escn	escn	nop	nop	O	nop
14	bs	bs	bs	bs	bs	bs	bs	bs	O	nop
15	ht	btab	ht	btab	ht	btab	ht	btab	O	nop
16	'q'	'Q'	dc1	dc1	escn	escn	nop	nop	C	nop
17	'w'	'W'	etb	etb	escn	escn	nop	nop	C	nop
18	'e'	'E'	enq	enq	escn	escn	nop	nop	C	nop
19	'r'	'R'	dc2	dc2	escn	escn	nop	nop	C	nop
20	't'	'T'	dc4	dc4	escn	escn	nop	nop	C	nop
21	'y'	'Y'	em	em	escn	escn	nop	nop	C	nop
22	'u'	'U'	nak	nak	escn	escn	nop	nop	C	nop
23	'i'	'I'	ht	ht	escn	escn	nop	nop	C	nop
24	'o'	'O'	si	si	escn	escn	nop	nop	C	nop
25	'p'	'P'	dle	dle	escn	escn	nop	nop	C	K_PREV
26	'['	'{'	esc	nop	escn	escn	nop	nop	O	nop
27	']'	'}'	gs	nop	escn	escn	nop	nop	O	nop
28	cr	cr	cr	cr	cr	cr	cr	cr	O	nop
29	lctrl	lctrl	lctrl	lctrl	lctrl	lctrl	lctrl	lctrl	O	nop
30	'a'	'A'	soh	soh	escn	escn	nop	nop	C	nop
31	's'	'S'	dc3	dc3	escn	escn	nop	nop	C	nop
32	'd'	'D'	eot	eot	escn	escn	k_dbg	nop	C	nop
33	'f'	'F'	ack	ack	escn	escn	nop	nop	C	K_FRCNEXT
34	'g'	'G'	bel	bel	escn	escn	nop	nop	C	nop
35	'h'	'H'	bs	bs	escn	escn	nop	nop	C	K_VTF
36	'j'	'J'	nl	nl	escn	escn	nop	nop	C	nop
37	'k'	'K'	vt	vt	escn	escn	nop	nop	C	nop
38	'l'	'L'	np	np	escn	escn	nop	nop	C	nop
39	';'	':'	';'	':'	escn	escn	nop	nop	O	nop
40	'''	'"'	'''	'"'	escn	escn	nop	nop	O	nop
41	'`'	'~'	'`'	'~'	escn	escn	nop	nop	O	nop
42	lshift	lshift	lshift	lshift	lshift	lshift	lshift	lshift	O	nop
43	'\\'	'|'	fs	'|'	escn	escn	nop	nop	O	nop
44	'z'	'Z'	sub	sub	escn	escn	nop	nop	C	nop

SCAN CODE	BASE	SHIFT	CTRL	CTRL SHIFT	ALT	ALT SHIFT	ALT CTRL	ALT CTRL SHIFT	LOCK	SRQTAB
45	'x'	'X'	can	can	escn	escn	nop	nop	C	nop
46	'c'	'C'	etx	etx	escn	escn	nop	nop	C	nop
47	'v'	'V'	syn	syn	escn	escn	nop	nop	C	nop
48	'b'	'B'	stx	stx	escn	escn	nop	nop	C	nop
49	'n'	'N'	so	so	escn	escn	nop	nop	C	K_NEXT
50	'm'	'M'	cr	cr	escn	escn	nop	nop	C	nop
51	','	'<'	','	'<'	escn	escn	nop	nop	O	nop
52	'.'	'>'	'.'	'>'	escn	escn	nop	nop	O	nop
53	'/'	'?'	'/'	ns	escn	escn	nop	nop	O	nop
54	rshift	rshift	rshift	rshift	rshift	rshift	rshift	rshift	O	nop
55	'*'	'*'	'*'	'*'	escn	escn	nop	nop	O	nop
56	lalt	lalt	lalt	lalt	lalt	lalt	lalt	lalt	O	nop
57	' '	' '	nul	nul	escn	escn	nop	nop	O	nop
58	clock	clock	clock	clock	clock	clock	clock	clock	O	nop
59	fkey1	fkey13	fkey25	fkey37	fkey1	fkey13	fkey25	fkey37	O	K_VTF+1
60	fkey2	fkey14	fkey26	fkey38	fkey2	fkey14	fkey26	fkey38	O	K_VTF+2
61	fkey3	fkey15	fkey27	fkey39	fkey3	fkey15	fkey27	fkey39	O	K_VTF+3
62	fkey4	fkey16	fkey28	fkey40	fkey4	fkey16	fkey28	fkey40	O	K_VTF+4
63	fkey5	fkey17	fkey29	fkey41	fkey5	fkey17	fkey29	fkey41	O	K_VTF+5
64	fkey6	fkey18	fkey30	fkey42	fkey6	fkey18	fkey30	fkey42	O	K_VTF+6
65	fkey7	fkey19	fkey31	fkey43	fkey7	fkey19	fkey31	fkey43	O	K_VTF+7
66	fkey8	fkey20	fkey32	fkey44	fkey8	fkey20	fkey32	fkey44	O	K_VTF+8
67	fkey9	fkey21	fkey33	fkey45	fkey9	fkey21	fkey33	fkey45	O	K_VTF+9
68	fkey10	fkey22	fkey34	fkey46	fkey10	fkey22	fkey34	fkey46	O	K_VTF+10
69	nlock	nlock	nlock	nlock	nlock	nlock	nlock	nlock	O	
70	slock	slock	slock	slock	slock	slock	slock	slock	O	
71	fkey49	'7'	fkey49	'7'	fkey49	escn	nop	nop	N	
72	fkey50	'8'	fkey50	'8'	fkey50	escn	nop	nop	N	
73	fkey51	'9'	fkey51	'9'	fkey51	escn	nop	nop	N	
74	fkey52	'-'	fkey52	'-'	fkey52	escn	nop	nop	N	
75	fkey53	'4'	fkey53	'4'	fkey53	escn	nop	nop	N	
76	fkey54	'5'	fkey54	'5'	fkey54	escn	nop	nop	N	
77	fkey55	'6'	fkey55	'6'	fkey55	escn	nop	nop	N	
78	fkey56	'+'	fkey56	'+'	fkey56	escn	nop	nop	N	
79	fkey57	'1'	fkey57	'1'	fkey57	escn	nop	nop	N	
80	fkey58	'2'	fkey58	'2'	fkey58	escn	nop	nop	N	
81	fkey59	'3'	fkey59	'3'	fkey59	escn	nop	nop	N	
82	fkey60	'0'	fkey60	'0'	fkey60	escn	nop	nop	N	
83	del	'.'	del	'.'	del	escn	rboot	nop	N	
84	fkey60	fkey26	fkey60	nop	sysreq	sysreq	sysreq	sysreq	O	
85	fkey58	fkey58	fkey58	fkey58	fkey58	fkey58	fkey58	fkey58	O	
86	fkey53	fkey53	fkey53	fkey53	fkey53	fkey53	fkey53	fkey53	O	
87	fkey11	fkey23	fkey35	fkey47	fkey11	fkey23	fkey35	fkey47	O	K_VTF+11
88	fkey12	fkey24	fkey36	fkey48	fkey12	fkey24	fkey36	fkey48	O	K_VTF+12

SCAN CODE	BASE	SHIFT	CTRL	CTRL SHIFT	ALT	ALT SHIFT	ALT CTRL	ALT CTRL SHIFT	LOCK	SRQTAB
89	nop	nop	nop	nop	nop	nop	nop	nop	O	K_NOP
90	nop	nop	nop	nop	nop	nop	nop	nop	O	K_NOP
91	nop	nop	nop	nop	nop	nop	nop	nop	O	K_NOP
92	nop	nop	nop	nop	nop	nop	nop	nop	O	K_NOP
93	nop	nop	nop	nop	nop	nop	nop	nop	O	K_NOP
94	nop	nop	nop	nop	nop	nop	nop	nop	O	K_NOP
95	nop	nop	nop	nop	nop	nop	nop	nop	O	K_NOP
96	nop	nop	nop	nop	nop	nop	nop	nop	O	K_NOP
97	nop	nop	nop	nop	nop	nop	nop	nop	O	K_NOP
98	nop	nop	nop	nop	nop	nop	nop	nop	O	K_NOP
99	nop	nop	nop	nop	nop	nop	nop	nop	O	K_NOP
100	nop	nop	nop	nop	nop	nop	nop	nop	O	K_NOP
101	nop	nop	nop	nop	nop	nop	nop	nop	O	K_NOP
102	nop	nop	nop	nop	nop	nop	nop	nop	O	K_NOP
103	nop	nop	nop	nop	nop	nop	nop	nop	O	K_NOP
104	nop	nop	nop	nop	nop	nop	nop	nop	O	K_NOP
105	nop	nop	nop	nop	nop	nop	nop	nop	O	K_NOP
106	nop	nop	nop	nop	nop	nop	nop	nop	O	K_NOP
107	fkey53	fkey53	fkey53	fkey53	fkey53	fkey53	fkey53	fkey53	O	
108	nop	nop	nop	nop	nop	nop	nop	nop	O	K_NOP
109	nop	nop	nop	nop	nop	nop	nop	nop	O	K_NOP
110	nop	nop	nop	nop	nop	nop	nop	nop	O	K_NOP
111	fkey51	fkey51	nop	nop	nop	nop	nop	nop	O	K_NOP
112	nop	nop	nop	nop	nop	nop	nop	nop	O	K_NOP
113	nop	nop	nop	nop	nop	nop	nop	nop	O	K_NOP
114	ralt	ralt	ralt	ralt	ralt	ralt	ralt	ralt	O	K_NOP
115	rctrl	rctrl	rctrl	rctrl	rctrl	rctrl	rctrl	rctrl	O	K_NOP
116	slock	slock	brk	brk	slock	slock	brk	brk	O	nline
117	'/'	'/'	nop	nop	escn	escn	nop	nop	O	K_NOP
118	nop	nop	nop	nop	nop	nop	nop	nop	O	K_NOP
119	slock	slock	brk	brk	slock	slock	brk	brk	O	K_NOP
120	fkey50	fkey50	nop	nop	nop	nop	nop	nop	O	K_NOP
121	del	del	del	del	del	del	rboot	del	O	K_NOP
122	fkey57	fkey57	nop	nop	nop	nop	nop	nop	O	K_NOP
123	fkey60	fkey60	nop	nop	nop	nop	nop	nop	O	K_NOP
124	nop	nop	nop	nop	nop	nop	nop	nop	O	K_NOP
125	fkey55	fkey55	nop	nop	nop	nop	nop	nop	O	K_NOP
126	fkey59	fkey59	nop	nop	nop	nop	nop	nop	O	K_NOP
127	fkey49	fkey49	nop	nop	nop	nop	nop	nop	O	K_NOP

The following table lists the value of each of the special keywords used in the preceding tables. The keywords are used only in the preceding tables for readability. In the actual keyboard map, a special keyword is represented by its value with the corresponding special state bit being set.

Name	Value	Meaning
nop	0	No operation - no action from keypress
lshift	2	Left-hand shift
rshift	3	Right-hand shift
clock	4	Caps lock
nlock	5	Numeric lock
slock	6	Scroll lock
alt	7	Alt key
btab	8	Back tab key - generates fixed sequence (ESC[Z)
ctrl	9	Control key
lalt	10	Left-hand alt key
ralt	11	right-hand alt key
lctrl	12	Left-hand control key
rctrl	13	Right-hand control key
agr	14	ALT-GR key (European keyboards only)
fkey1	27	Function key #1
.		.
.		.
.		.
fkey96	122	Function key #96
sysreq	123	System request
brk	124	Break key
escn	125	Generate an ESC N *x* sequence, where *x* is the un-alt'ed value of the scan code
esco	126	Generate an ESC O *x* sequence, where *x* is the un-alt'ed value of the scan code
escl	127	Generate an ESC L *x* sequence, where *x* is the un-alt'ed value of the scan code
rboot	128	Reboot system
debug	129	Invoke kernel debugger
NEXT	130	Switch to next virtual terminal on queue
PREV	131	Switch to previous virtual terminal on queue
FNEXT	132	Forced switch to next virtual terminal on queue
FPREV	133	Forced switch to previous virtual terminal on queue
VTF	134	Virtual Terminal First (VT00)
.		.
.		.
.		.
VTL	148	Virtual Terminal Last (VT14)
MGRF	149	Virtual Terminal Manager First. Allows assigning special significance to key sequence for actions by virtual terminal layer manager. Used in SRQTAB table.
.		.
.		.
.		.
MGRL	179	Virtual Terminal Manager Last. Used in SRQTAB table.

The following table lists names and decimal values for ASCII characters in the preceding table. Names are used in place of numeric constants to make it easier to read the scan code table. Only the decimal values are placed in the **ioctl** buffer. These values are taken from **ascii**(5).

Name	Value	Name	Value
nul	0	dc1	17
soh	1	dc2	18
stx	2	dc3	19
etx	3	dc4	20
eot	4	nak	21
enq	5	syn	22
ack	6	etb	23
bel	7	can	24
bs	8	em	25
ht	9	sub	26
nl	10	esc	27
vt	11	fs	28
np	12	gs	29
cr	13	rs	30
so	14	ns	31
si	15	del	127
dle	16		

String Key Mapping

The string mapping table is an array of 512 bytes (typedef **strmap_t**) containing null-terminated strings that redefine the function keys. The first null-terminated string is assigned to the first function key, the second string is assigned to the second function key, and so on.

There is no limit to the length of any particular string; however, the whole table can not exceed 512 bytes, including nulls. To make a string a null, add extra null characters. The following table contains default function key values.

Default Function Key Values

Function Key #	Function	Shift Function	Ctrl Function	Ctrl Shift Function
1	ESC OP	ESC Op	ESC OP	ESC Op
2	ESC OQ	ESC Oq	ESC OQ	ESC Oq
3	ESC OR	ESC Or	ESC OR	ESC Or
4	ESC OS	ESC Os	ESC OS	ESC Os
5	ESC OT	ESC Ot	ESC OT	ESC Ot
6	ESC OU	ESC Ou	ESC OU	ESC Ou
7	ESC OV	ESC Ov	ESC OV	ESC Ov
8	ESC OW	ESC Ow	ESC OW	ESC Ow
9	ESC OX	ESC Ox	ESC OX	ESC Ox
10	ESC OY	ESC Oy	ESC OY	ESC Oy
11	ESC OZ	ESC Oz	ESC OZ	ESC Oz
12	ESC OA	ESC Oa	ESC OA	ESC Oa

Ioctl Calls:

KDGKBMODE

This call gets the current keyboard mode. It returns one of the following values, as defined in **/usr/include/sys/kd.h**:

```
#define K_RAW        0x00    /* Send row scan codes */
#define K_XLATE      0x01    /* Translate to ASCII */
```

KDSKBMODE

This call sets the keyboard mode. The argument to the call is either **K_RAW** or **K_XLATE**. By using raw mode, the program can see the raw up and down scan codes from the keyboard. In translate mode, the translation tables are used to generate the appropriate character code.

KDGKBTYPE

This call gets the keyboard type. It returns one of the following values, as defined in **/usr/include/sys/kd.h**:

```
#define KB_84        1       /*84 key keyboard*/
#define KB_101       2       /*101 key keyboard*/
#define KB_OTHER     3       /*Other type keyboard*/
```

KDGKBENT

This call reads one of the entries in the translation tables. The argument to the call is the address of one of the following structures, defined in **/usr/include/sys/kd.h**, with the first two fields filled in:

```
struct kbentry {
        unchar kb_table;       /* Table to use */
        unchar kb_index;       /* Entry in table */
        ushort kb_value;       /* Value to get/set */
};
```

Valid values for the *kb_table* field are:

```
#define K_NORMTAB        0x00    /* Base */
#define K_SHIFTTAB       0x01    /* Shifted */
#define K_ALTTAB         0x02    /* Alt */
#define K_ALTSHIFTTAB    0x03    /* Shifted alt */
#define K_SRQTAB         0x04    /* Select sysreq
                                         table */
```

The **ioctl** will get the indicated entry from the indicated table and return it in the third field.

The **K_SRQTAB** value for the kb_table field allows access to the scancode indexed table which allows assignment of a given virtual terminal selector (**K_VTF–K_VTL**) or the virtual terminal layer manager (**K_MGRF–K_MGRL**) specialkey assignments.

The virtual terminal selector (**K_VTF**) is normally associated with **/dev/tty00**, on which the user login shell is commonly found. The following terminal selectors also are used to select virtual terminals:

K_VTF+1 for the 1st virtual terminal (/dev/vt01)
K_VTF+2 for the 2nd virtual terminal (/dev/vt02)

.
.
.

K_VTF+12 for the 12th virtual terminal (/dev/vt12)

KDSKBENT

This call sets an entry in one of the translation tables. It uses the same structure as the **KDGKBENT ioctl**, but with the third field filled in with the value that should be placed in the translation table. This can be used to partially or completely remap the keyboard. This ioctl does not work for all keycodes.

The **kd** driver provides support for virtual terminals. The console minor device, **/dev/vtmon**, provides virtual terminal key requests from the **kd** driver to the process that has **/dev/vtmon** open. Two **ioctls** are provided for virtual terminal support:

VT_GETSTATE

The **VT_GETSTATE ioctl** returns global virtual terminal state information. It returns the active virtual terminal in the v_active field, and the number of active virtual terminals and a bit mask of the global state in the v_state open field, where bit x is the state of vt x (1 indicates that the virtual terminal is open).

VT_SENDSIG

The **VT_SENDSIG ioctl** specifies a signal (in v_signal) to be sent to a bit mask of virtual terminals (in v_state).

The data structure used by the **VT_GETSTATE** and **VT_SENDSIG** **ioctl**s is:

```
struct vt_stat {
        ushort v_active;   /* active vt */
        ushort v_signal;   /* signal to send (VT_SENDSIG) */
        ushort v_state;    /* vt bit mask (VT_SENDSIG and VT_GETSTATE) */
};
```

and is defined in **/usr/include/sys/vt.h** .

VT_OPENQRY
: The **VT_OPENQRY** **ioctl** is used to get the next available virtual terminal. It inquires if this vt is already open. This value is set in the last argument of the **ioctl** (2) call.

GIO_KEYMAP
: This call gets the entire keyboard mapping table from the kernel. The structure of the argument is given in **/usr/include/sys/kd.h**.

PIO_KEYMAP
: This call sets the entire keyboard mapping table. The structure of the argument is given in **/usr/include/sys/kd.h**.

GIO_STRMAP
: This call gets the function key string mapping table from the kernel. The structure of the argument is given in **/usr/include/sys/kd.h**.

PIO_STRMAP
: This call sets the function key string mapping table. The structure of the argument is given in **/usr/include/sys/kd.h**.

TIOCKBOF
: Extended character codes are disabled. This is the default mode.

TIOCKBON
: Allows extended characters to be transmitted to the user program when the extended keys are enabled. Then the keyboard is said to be fully enabled. The extended characters are transmitted as a null byte followed by a second byte containing the character's extended code. When a true null byte is sent, it is transmitted as two consecutive null bytes.

When the keyboard is fully enabled, an 8-bit character code can be obtained by holding down the **ALT** key and entering the 3-digit decimal value of the character from the numeric keypad. The character is transmitted when the **ALT** key is released.

Some keyboard characters have special meaning. Under default operations, pressing the **DELETE** key generates an interrupt signal that is sent to all processes designated with the associated control terminal. When the keyboard is fully enabled, holding down the **ALT** key and pressing the 8 key on the home keyboard (not on the numeric keypad) returns a null byte followed by 0x7F. This will produce the same effect as the **DELETE** key (0x7F) unless you have executed the **stty**(1) command with the **-isig** option.

KBENABLED

If the keyboard is fully enabled (**TIOCKBON**), a non-zero value will be returned. If the keyboard is not fully enabled (**TIOCKBOF**), a value of zero will be returned.

GETFKEY

Obtains the current definition of a function key. The argument to the call is the address of one of the following structures defined in **/usr/include/sys/kd.h**:

```
struct fkeyarg {
    unsigned short keynum;
    unchar   keydef [MAXFK];      /*Comes from ioctl.h via comcrt.h*/
    char   flen;
};
```

The function key number must be passed in *keynum* (see *arg* structure above). The string currently assigned to the key will be returned in *keydef* and the length of the string will be returned in *flen* when the **ioctl** is performed.

SETFKEY

Assigns a given string to a function key. It uses the same structure as the **GETFKEY ioctl**. The function key number must be passed in *keynum*, the string must be passed in *keydef*, and the length of the string (number of characters) must be passed in *flen*.

FILES

/dev/console
/dev/vt00-n
/usr/include/sys/kd.h

SEE ALSO

ascii(5), **console**(7), **display**(7), **ioctl**(2), **stty**(1), **termio**(7), **vtlmgr**(1)

NAME

kmem – perform I/O on kernel memory based on symbol name

SYNOPSIS

```
#include <sys/ksym.h>

int ioctl(int kmemfd, MIOC_READKSYM, struct mioc_rksym *rks);
int ioctl(int kmemfd, MIOC_IREADKSYM, struct mioc_rksym *rks);
int ioctl(int kmemfd, MIOC_WRITEKSYM, struct mioc_rksym *rks);
```

DESCRIPTION

When used with a valid file descriptor for `/dev/kmem` (*kmemfd*), these ioctl commands [see `ioctl`(2)] can be used to read or write kernel memory based on information provided in the `mioc_rksym` structure, which includes the following members:

```
char *mirk_symname;     /* symbol at whose address read will start */
void *mirk_buf;         /* buffer into which data will be written */
size_t mirk_buflen;     /* length (in bytes) of read buffer */
```

The second argument to `ioctl` determines which I/O operation is being performed:

`ioctl`	Meaning
`MIOC_READKSYM`	Read *buflen* bytes of kernel memory starting at the address for *symname* into *buf.*
`MIOC_IREADKSYM`	Read `sizeof(void *)` bytes of kernel memory starting at the address for *symname* and use that as the address from which to read *buflen* bytes of kernel memory into *buf.*
`MIOC_WRITEKSYM`	Write *buflen* bytes into kernel memory starting at the address for *symname* from *buf.*

DIAGNOSTICS

In addition to the error conditions listed on `ioctl`(2), these `ioctl` commands can fail for the following reasons:

`EBADF`	*kmemfd* open for reading and this is `MIOC_WRITEKSYM` or *kmemfd* open for writing and this is `MIOC_READKSYM`
`EFAULT`	Value of `mirk_buflen` results in attempt to read outside kernel virtual address space, or the third argument to `ioctl` is an invalid pointer, or an invalid pointer is given for the symbol name or buffer in the `mioc_rksym` structure
`EINVAL`	Second argument to `ioctl` is invalid
`ENAMETOOLONG`	Symbol name is longer than `MAXSYMNMLEN` characters
`ENOMATCH`	Symbol names not found in the running kernel (including loaded modules)
`ENXIO`	*kmemfd* open on wrong minor device (that is, not `/dev/kmem`)

SEE ALSO

`getksym`(2), `ioctl`(2), `nlist`(3E)

NAME

ldterm – standard STREAMS terminal line discipline module

DESCRIPTION

ldterm is a STREAMS module that provides most of the **termio**(7) terminal interface. This module does not perform the low-level device control functions specified by flags in the **c_cflag** word of the **termio/termios** structure or by the **IGNBRK**, **IGNPAR**, **PARMRK**, or **INPCK** flags in the **c_iflag** word of the **termio/termios** structure; those functions must be performed by the driver or by modules pushed below the **ldterm** module. All other **termio/termios** functions are performed by **ldterm**; some of them, however, require the cooperation of the driver or modules pushed below **ldterm** and may not be performed in some cases. These include the **IXOFF** flag in the **c_iflag** word and the delays specified in the **c_oflag** word.

ldterm also handles EUC and multi-byte characters.

When **ldterm** is pushed onto a stream, the **open** routine initializes the settings of the **termio** flags. The default settings are:

```
c_iflag = BRKINT|ICRNL|IXON|ISTRIP
c_oflag = OPOST|ONLCR|TAB3
c_cflag = 0
c_lflag = ISIG|ICANON|ECHO|ECHOK
```

The remainder of this section describes the processing of various STREAMS messages on the read- and write-side.

Read-side Behavior

Various types of STREAMS messages are processed as follows:

M_BREAK When this message is received, either an interrupt signal is generated or the message is treated as if it were an **M_DATA** message containing a single ASCII NUL character, depending on the state of the **BRKINT** flag.

M_DATA This message is normally processed using the standard **termio** input processing. If the **ICANON** flag is set, a single input record (''line'') is accumulated in an internal buffer and sent upstream when a line-terminating character is received. If the **ICANON** flag is not set, other input processing is performed and the processed data are passed upstream.

If output is to be stopped or started as a result of the arrival of characters (usually **CNTRL-Q** and **CNTRL-S**), **M_STOP** and **M_START** messages are sent downstream. If the **IXOFF** flag is set and input is to be stopped or started as a result of flow-control considerations, **M_STOPI** and **M_STARTI** messages are sent downstream.

M_DATA messages are sent downstream, as necessary, to perform echoing.

If a signal is to be generated, an **M_FLUSH** message with a flag byte of **FLUSHR** is placed on the read queue. If the signal is also to flush output, an **M_FLUSH** message with a flag byte of **FLUSHW** is sent downstream.

M_CTL If the size of the data buffer associated with the message is the size of **struct iocblk, ldterm** will perform functional negotiation to determine where the **termio**(7) processing is to be done. If the command field of the **iocblk** structure (**ioc_cmd**) is set to **MC_NO_CANON**, the input canonical processing normally performed on **M_DATA** messages is disabled and those messages are passed upstream unmodified; this is for the use of modules or drivers that perform their own input processing, such as a pseudo-terminal in **TIOCREMOTE** mode connected to a program that performs this processing. If the command is **MC_DO_CANON**, all input processing is enabled. If the command is **MC_PART_CANON**, then an **M_DATA** message containing a **termios** structure is expected to be attached to the original **M_CTL** message. The **ldterm** module will examine the **iflag, oflag**, and **lflag** fields of the **termios** structure and from then on will process only those flags which have not been turned ON. If none of the above commands are found, the message is ignored; in any case, the message is passed upstream.

M_FLUSH The read queue of the module is flushed of all its data messages and all data in the record being accumulated are also flushed. The message is passed upstream.

M_IOCACK The data contained within the message, which is to be returned to the process, are augmented if necessary, and the message is passed upstream.

All other messages are passed upstream unchanged.

Write-side Behavior

Various types of STREAMS messages are processed as follows:

M_FLUSH The write queue of the module is flushed of all its data messages and the message is passed downstream.

M_IOCTL The function of this **ioctl** is performed and the message is passed downstream in most cases. The **TCFLSH** and **TCXONC ioctl**s can be performed entirely in the **ldterm** module, so the reply is sent upstream and the message is not passed downstream.

M_DATA If the **OPOST** flag is set, or both the **XCASE** and **ICANON** flags are set, output processing is performed and the processed message is passed downstream along with any **M_DELAY** messages generated. Otherwise, the message is passed downstream without change.

All other messages are passed downstream unchanged.

ioctls

The following **ioctl**s are processed by the **ldterm** module. All others are passed downstream. **EUC_WSET** and **EUC_WGET** are **I_STR ioctl** calls whereas other **ioctl**s listed here are **TRANSPARENT ioctl**s.

TCGETS/TCGETA
The message is passed downstream; if an acknowledgment is seen, the data provided by the driver and modules downstream are augmented and the acknowledgement is passed upstream.

TCSETS/TCSETSW/TCSETSF/TCSETA/TCSETAW/TCSETAF
The parameters that control the behavior of the **ldterm** module are changed. If a mode change requires options at the stream head to be changed, an **M_SETOPTS** message is sent upstream. If the **ICANON** flag is turned on or off, the read mode at the stream head is changed to message-nondiscard or byte-stream mode, respectively. If the **TOSTOP** flag is turned on or off, the tostop mode at the stream head is turned on or off, respectively.

TCFLSH If the argument is 0, an **M_FLUSH** message with a flag byte of **FLUSHR** is sent downstream and placed on the read queue. If the argument is 1, the write queue is flushed of all its data messages and an **M_FLUSH** message with a flag byte of **FLUSHW** is sent upstream and downstream. If the argument is 2, the write queue is flushed of all its data messages and an **M_FLUSH** message with a flag byte of **FLUSHRW** is sent downstream and placed on the read queue.

TCXONC If the argument is 0 and output is not already stopped, an **M_STOP** message is sent downstream. If the argument is 1 and output is stopped, an **M_START** message is sent downstream. If the argument is 2 and input is not already stopped, an **M_STOPI** message is sent downstream. If the argument is 3 and input is stopped, an **M_STARTI** message is sent downstream.

TCSBRK The message is passed downstream, so the driver has a chance to drain the data and then send and an **M_IOCACK** message upstream.

EUC_WSET This call takes a pointer to an **eucioc** structure, and uses it to set the EUC line discipline's local definition for the code set widths to be used for subsequent operations. Within the stream, the line discipline may optionally notify other modules of this setting via **M_CTL** messages.

EUC_WGET This call takes a pointer to an **eucioc** structure, and returns in it the EUC code set widths currently in use by the EUC line discipline.

SEE ALSO

pseudo(1), **console**(7), **termio**(7), **termios**(2)

NAME

`lo` – software loopback network interface

SYNOPSIS

```
d = open ("/dev/loop", O_RDWR);
```

DESCRIPTION

The **`loopback`** device is a software datalink provider (interface driver) that returns all packets it receives to their source without involving any hardware devices. It is a STREAMS device conforming to the datalink provider interface (DLPI). See **`if`**(7) for a general description of network interfaces.

The **`loopback`** interface is used to access Internet services on the local machine. Because it is available on all machines, including those with no hardware network interfaces, programs can use it for guaranteed access to local servers. A typical application is the **`comsat`**(1M) server which accepts notification of mail delivery from a local client. The loopback interface is also used for performance analysis and testing.

By convention, the name of the loopback interface is `lo0`, and it is configured with Internet address 127.0.0.1. This address may be changed with the **`SIOCSIFADDR`** **`ioctl()`**.

SEE ALSO

`comsat`(1M), **`if`**(7), **`inet`**(7)

NAME

log – interface to STREAMS error logging and event tracing

SYNOPSIS

```
#include <sys/stream.h>
#include <sys/log.h>
#include <sys/strlog.h>
#include <sys/syslog.h>
```

DESCRIPTION

log is a STREAMS software device driver that provides an interface for console logging and for the STREAMS error logging and event tracing processes [**strerr**(1M), **strace**(1M)]. **log** presents two separate interfaces: a function call interface in the kernel through which STREAMS drivers and modules submit **log** messages; and a subset of **ioctl**(2) system calls and STREAMS messages for interaction with a user level console logger, an error logger, a trace logger, or processes that need to submit their own **log** messages.

Kernel Interface

log messages are generated within the kernel by calls to the function **strlog**:

```
strlog(short mid, short sid, char level, ushort flags,
     char *fmt, unsigned arg1, ... );
```

Required definitions are contained in **stream.h**, **strlog.h**, **log.h**, and **syslog.h** in **/usr/include/sys**. *mid* is the STREAMS module ID number for the module or driver submitting the **log** message. *sid* is an internal sub-ID number usually used to identify a particular minor device of a driver. *level* is a tracing level that allows for selective screening out of low priority messages from the tracer. *flags* are any combination of **SL_ERROR** (the message is for the error logger), **SL_TRACE** (the message is for the tracer), **SL_CONSOLE** (the message is for the console logger), **SL_FATAL** (advisory notification of a fatal error), and **SL_NOTIFY** (request that a copy of the message be mailed to the system administrator). *fmt* is a **printf**(3S) style format string, except that **%s**, **%e**, **%E**, **%g**, and **%G** conversion specifications are not handled. Up to **NLOGARGS** (currently 3) numeric or character arguments can be provided.

User Interface

log is opened via the **clone** interface, **/dev/log**. Each open of **/dev/log** obtains a separate stream to **log**. In order to receive **log** messages, a process must first notify **log** whether it is an error logger, trace logger, or console logger via a STREAMS **I_STR ioctl** call (see below). For the console logger, the **I_STR ioctl** has an **ic_cmd** field of **I_CONSLOG**, with no accompanying data. For the error logger, the **I_STR ioctl** has an **ic_cmd** field of **I_ERRLOG**, with no accompanying data. For the trace logger, the **ioctl** has an **ic_cmd** field of **I_TRCLOG**, and must be accompanied by a data buffer containing an array of one or more struct **trace_ids** elements. Each **trace_ids** structure specifies an *mid*, *sid*, and *level* from which message will be accepted. **strlog** will accept messages whose *mid* and *sid* exactly match those in the **trace_ids** structure, and whose level is less than or equal to the level given in the **trace_ids** structure. A value of –1 in any of the fields of the **trace_ids** structure indicates that any value is accepted for that field.

Once the logger process has identified itself via the **ioctl** call, **log** will begin sending up messages subject to the restrictions noted above. These messages are obtained via the **getmsg(2)** system call. The control part of this message contains a **log_ctl** structure, which specifies the *mid, sid, level, flags,* time in ticks since boot that the message was submitted, the corresponding time in seconds since Jan. 1, 1970, a sequence number, and a priority. The time in seconds since 1970 is provided so that the date and time of the message can be easily computed, and the time in ticks since boot is provided so that the relative timing of **log** messages can be determined.

The priority is comprised of a priority code and a facility code, found in **sys/syslog.h**. If **SL_CONSOLE** is set in *flags,* the priority code is set as follows. If **SL_WARN** is set, the priority code is set to **LOG_WARNING**. If **SL_FATAL** is set, the priority code is set to **LOG_CRIT**. If **SL_ERROR** is set, the priority code is set to **LOG_ERR**. If **SL_NOTE** is set, the priority code is set to **LOG_NOTICE**. If **SL_TRACE** is set, the priority code is set to **LOG_DEBUG**. If only **SL_CONSOLE** is set, the priority code is set to **LOG_INFO**. Messages originating from the kernel have the facility code set to **LOG_KERN**. Most messages originating from user processes will have the facility code set to **LOG_USER**.

Different sequence numbers are maintained for the error and trace logging streams, and are provided so that gaps in the sequence of messages can be determined (during times of high message traffic some messages may not be delivered by the logger to avoid hogging system resources). The data part of the message contains the unexpanded text of the format string (null terminated), followed by **NLOGARGS** words for the arguments to the format string, aligned on the first word boundary following the format string.

A process may also send a message of the same structure to **log**, even if it is not an error or trace logger. The only fields of the **log_ctl** structure in the control part of the message that are accepted are the *level, flags,* and *pri* fields; all other fields are filled in by **log** before being forwarded to the appropriate logger. The data portion must contain a null terminated format string, and any arguments (up to **NLOGARGS**) must be packed one word each, on the next word boundary following the end of the format string.

ENXIO is returned for **I_TRCLOG ioctl**s without any **trace_ids** structures, or for any unrecognized **I_STR ioctl** calls. Incorrectly formatted **log** messages sent to the driver by a user process are silently ignored (no error results).

Processes that wish to write a message to the console logger may direct their output to **/dev/conslog**, using either **write**(2) or **putmsg**(2).

EXAMPLES

Example of **I_ERRLOG** notification:

```
struct strioctl ioc;

ioc.ic_cmd = I_ERRLOG;
ioc.ic_timout = 0;    /* default timeout (15 secs.) */
ioc.ic_len = 0;
ioc.ic_dp = NULL;

ioctl(log, I_STR, &ioc);
```

Example of `I_TRCLOG` notification:

```
struct trace_ids tid[2];

tid[0].ti_mid = 2;
tid[0].ti_sid = 0;
tid[0].ti_level = 1;

tid[1].ti_mid = 1002;
tid[1].ti_sid = -1;   /* any sub-id will be allowed */
tid[1].ti_level = -1; /* any level will be allowed */

ioc.ic_cmd = I_TRCLOG;
ioc.ic_timout = 0;
ioc.ic_len = 2 * sizeof(struct trace_ids);
ioc.ic_dp = (char *)tid;

ioctl(log, I_STR, &ioc);
```

Example of submitting a `log` message (no arguments):

```
struct strbuf ctl, dat;
struct log_ctl lc;
char *message = "Don't forget to pick up some milk
                 on the way home";

ctl.len = ctl.maxlen = sizeof(lc);
ctl.buf = (char *)&lc;

dat.len = dat.maxlen = strlen(message);
dat.buf = message;

lc.level = 0;
lc.flags = SL_ERROR|SL_NOTIFY;

putmsg(log, &ctl, &dat, 0);
```

FILES

/dev/log
/dev/conslog

SEE ALSO

clone(7), getmsg(2), intro(2), putmsg(2), strace(1M), strerr(1M), write(2)

NOTES

The `log` driver high and low water marks are tunable via the master file.

NAME

`lp` – parallel port interface

DESCRIPTION

The parallel port (`lp`) driver supports both the primary (monochrome) and secondary parallel printer adapters simultaneously. Up to two printers are supported. If an adapter for a printer is not installed, an attempt to open it will fail. The close waits until all output is completed before returning to the user. The lp driver allows only one process at a time to write to the adapter. If it is already busy, an open for writing will return an error. However, the driver allows multiple opens to occur if they are read-only.

The parallel printer adapters are character devices. The minor device number corresponds to the primary or secondary parallel printer adapter. Thus, minor device 0 corresponds to the primary parallel printer adapter, while minor device 1 corresponds to the secondary adapter.

The parallel port behaves as described in **`termio`**(7).

FILES

`/dev/lp*`

SEE ALSO

`stty`(1), **`termio`**(7)

NAME

mcis – MCIS SCSI host adapter driver

DESCRIPTION

The MCIS host adapter subsystem serves as a means for SCSI target drivers (such as **sd01**, **st01**, and so on) to communicate on the SCSI bus with target controllers and logical units. This driver implements the SCSI Driver Interface (SDI) for such SCSI target drivers.

It is also possible to access this subsystem using the pass-through driver interface. To find the appropriate device to use, while any device is being accessed through the target driver, use the **B_GETDEV** ioctl to get the major and minor numbers of the pass-through node. This node may be created and opened for use.

ioctl Calls

There are three groups of **ioctl**(2) commands supported by **mcis**. The first group contains the ioctl commands used by the **mcis** driver itself:

SDI_SEND
: Sends a pass-through command to a target controller, bypassing the associated target driver.

SDI_BRESET
: Resets the SCSI bus.

B_REDT
: Reads the extended **edt** data structure that is stored in the **mcis** driver's internal data structure.

B_HA_CNT
: Gets the value of the number of host adapters for which the **mcis** driver is configured.

The second group is used by the **mcis** driver and all target drivers that use the SDI protocol to communicate with their associated target controllers.

B_GETTYPE
: Returns the bus name (for example, **scsi**) and device driver name of a specific device.

The third group should be supported by all target drivers that use the SDI protocol to communicate with their associated target controllers. However, this **ioctl** is not supported by the **mcis** driver.

B_GETDEV
: Returns the pass-through major and minor numbers to the calling utility to allow creation of a pass-through special device file.

Files

```
/usr/include/sys/mcis.h
/usr/include/sys/scsi.h
/usr/include/sys/sdi.h
/usr/include/sys/sdi_edt.h
/etc/conf/pack.d/mcis/space.c
```

NOTICES

On IBM MCA SCSI systems (for example, Model 57 and Model 90), the SCSI boot disk must be configured to be at SCSI target address (SCSI ID) 6. If you attempt to use a SCSI disk not at SCSI ID 6 as a boot device, it may not work.

REFERENCES

`ioctl`(2)

mem (7)

NAME

mem, kmem – core memory

DESCRIPTION

The file **/dev/mem** is a special file that is an image of the core memory of the computer. For example, it may be used to examine and patch the system.

Byte addresses in **/dev/mem** are interpreted as memory addresses. References to non-existent locations cause errors to be returned.

Examining and patching device registers is likely to lead to unexpected results when read-only or write-only bits are present.

The file **/dev/kmem** is the same as **/dev/mem** except that kernel virtual memory rather than physical memory is accessed.

Files

/dev/mem
/dev/kmem

NOTES

Some of **/dev/kmem** cannot be read because of write-only addresses or unequipped memory addresses.

NAME

mouse – mouse device driver for bus, serial, and PS/2 mouse devices

DESCRIPTION

The mouse device driver supports several types of mouse devices:

Logitech bus mouse that attaches to a plug-in card and is designed to be used in an eight-bit card slot.

Logitech serial type mouse that plugs directly into a serial port connector.

PS/2 compatible mouse that connects to a PS/2 auxiliary port.

Microsoft bus mouse that attaches to a plug-in card and is designed to be used in an eight-bit card slot.

Microsoft serial type mouse that plugs directly into a serial port connector.

The driver will support multiple mouse applications running in virtual terminal screens, both under the UNIX System and MS-DOS via SimulTask, VP/ix, Merge, or another similar product.

Support for mouse administration is also provided. See **mouseadmin**(1).

The following **ioctl**'s are supported:

MOUSEIOCMON Used exclusively by **/usr/lib/mousemgr** to receive open and close commands from **/dev/mouse** driver.

MOUSEISOPEN Used exclusively by **mouseadmin**. Returns 16-byte character array indicating which mouse devices are currently open; 1 is open, 0 is not open. The array is in the linear order established by **/usr/bin/mouseadmin** in building the display and device map pairs.

MOUSEIOCCONFIG
Used exclusively by **mouseadmin** to configure display and mouse pairs. The **mse_cfg** data structure is used to pass display and device mapping and map pair count information to the driver:

```
struct mse_cfg {
        struct mousemap *mapping;
        unsigned    count;
}
struct mousemap {
        dev_t    disp_dev;
        dev_t    mse_dev;
}
        int      type;
```

MOUSEIOCREAD Read mouse position and status data. The following data structure is used to return mouse position information to a user application:

```
struct mouseinfo {
       unsigned char status;
       char          xmotion:
       char          ymotion;
}
```

MOUSEIOCREAD will set **errno** to **EFAULT** for failure to return a valid **mouseinfo** structure. The **status** byte contains the button state information according to the following format:

```
0 Mv Lc Mc Rc L M R
```

where:

Mv:	is 1 if the mouse has moved since last **MOUSEIOCREAD**
Lc:	is 1 if Left button has changed state since last **MOUSEIOCREAD**
Mc:	is 1 if Middle button has changed state since last **MOUSEIOCREAD**
Rc:	is 1 if Right button has changed state since last **MOUSEIOCREAD**
L:	current state of Left button (1 == depressed)
M:	current state of Middle button
R:	current state of Right button

The **Mv** bit is required because the total x and y delta since the last **MOUSEIOCREAD** ioctl could be 0 yet the mouse may have been moved. The **Lc**, **Mc**, and **Rc** bits are required for a similar reason; if a button had been pushed and released since the last **MOUSEIOCREAD** ioctl, the current state bit would be unchanged but the application would want to know the button had been pushed.

The **xmotion** and **ymotion** fields are signed quantities relative to the previous position in the range -127 to 127. Deltas that would overflow a signed char have been truncated.

MOUSE320 Used to send commands and receive responses from the PS/2 compatible mouse devices. Failed **MOUSE320** commands will return **ENXIO** as the **errno** value. The following data structure is used to pass commands, status, and position information between the driver and a user application:

```
struct cmd_320 {
       int      cmd;
       int      arg1;
       int      arg2;
       int      arg3;
}
```

Valid commands for the PS/2 compatible devices are as follows:

MSERESET	reset mouse
MSERESEND	resend last data
MSESETDEF	set default status
MSEOFF	disable mouse
MSEON	enable mouse
MSESPROMPT	set prompt mode
MSEECHON	set echo mode
MSEECHOFF	reset echo mode
MSESTREAM	set stream mode
MSESETRES	set resolution (counts per millimeter) valid arg1 values are as follows: 00 = 1 count/mm. 01 = 2 count/mm. 02 = 4 count/mm. 03 = 8 count/mm.
MSESCALE2	set 2:1 scaling
MSESCALE1	set 1:1 scaling
MSECHGMOD	set sampling rate (reports per second) valid arg1 values are as follows: 0A = 10 reports/sec. 14 = 20 reports/sec. 28 = 40 reports/sec. 3C = 60 reports/sec. 50 = 80 reports/sec. 64 = 100 reports/sec. C8 = 200 reports/sec.
MSEGETDEV	read device type returns a zero (0) for the PS/2 compatible mouse.
MSEREPORT	read mouse report returns three-byte mouse/button position where bytes two and three are 9-bit 2's complement relative motions with the 9th bit (sign bit) coming from byte 1. Byte 1 b0 - left button (1 == depressed) b1 - right button b2 - middle button b3 - always 1 b4 - X data sign (1 == negative) b5 - Y data sign b6 - X data overflow b7 - Y data overflow

Byte 2
X axis position data

Byte 3
Y axis position data

MSESTATREQ status request returns three-byte report with the following format:

Byte 1
b0 - right button (1 == depressed)
b1 - middle button
b2 - left button
b3 - always 0
b4 - scaling 1:1 = 0, 2:1 = 1
b5 - disabled(0)/enabled(1)
b6 - stream(0)/prompt(1) mode
b7 - always 0

Byte 2
b0 - 6 current resolution
b7 - always 0

Byte 3
b0 - 7 current sampling rate

NOTE

The mouse also supports queue mode for accessing mouse input, both motion and button events; see **display**(7) for more information on the **KDQUEMODE ioctl**.

FILES

/dev/mouse
/usr/lib/mousemgr
/usr/include/sys/mouse.h

SEE ALSO

mouseadmin(1)

NAME

`null` – the null file

DESCRIPTION

Data written on the null special file, **`/dev/null`**, is discarded.

Reads from a null special file always return 0 bytes.

Files

`/dev/null`

NAME

pckt – STREAMS Packet Mode module

DESCRIPTION

pckt is a STREAMS module that may be used with a pseudo terminal to packetize certain messages. The **pckt** module should be pushed [see **I_PUSH**, **streamio**(7)] onto the master side of a pseudo terminal.

Packetizing is performed by prefixing a message with an **M_PROTO** message. The original message type is stored in the 4 byte data portion of the **M_PROTO** message.

On the read-side, only the **M_PROTO**, **M_PCPROTO**, **M_STOP**, **M_START**, **M_STOPI**, **M_STARTI**, **M_IOCTL**, **M_DATA**, **M_FLUSH**, and **M_READ** messages are packetized. All other message types are passed upstream unmodified.

Since all unread state information is held in the master's stream head read queue, flushing of this queue is disabled.

On the write-side, all messages are sent down unmodified.

With this module in place, all reads from the master side of the pseudo terminal should be performed with the **getmsg**(2) or **getpmsg** system call. The control part of the message contains the message type. The data part contains the actual data associated with that message type. The onus is on the application to separate the data into its component parts.

SEE ALSO

crash(1M), **getmsg**(2), **ioctl**(2), **ldterm**(7), **ptem**(7), **pty**(7), **streamio**(7), **termio**(7)

NAME

prf – operating system profiler

DESCRIPTION

The special file **/dev/prf** provides access to activity information in the operating system. The profiler commands load the measurement facility with text addresses to be monitored. Reading the file returns these addresses and a set of counters indicative of activity between adjacent text addresses.

The recording mechanism is driven by the system clock and samples the program counter at line frequency. Samples that catch the operating system are matched against the stored text addresses and increment corresponding counters for later processing.

The file **/dev/prf** is a pseudo-device with no associated hardware.

FILES

/dev/prf

NOTES

If the **prf** device is not configured into the kernel, to turn it on you must edit the **/etc/conf/sdevice.d/prf** file, change the second field from **N** to **Y**, and reconfigure the kernel.

When the profiler is turned on, loadable modules are locked into memory and cannot be unloaded. Subsequently loaded modules will also be locked until profiling is disabled.

SEE ALSO

profiler(1M)

NAME

ptem – STREAMS pseudo-terminal emulation module

DESCRIPTION

ptem is a STREAMS module that when used in conjunction with a line discipline and pseudo terminal driver emulates a terminal. See **pseudo**(1).

The **ptem** module must be pushed [see **I_PUSH, streamio**(7)] onto the slave side of a pseudo terminal STREAM, before the **ldterm** module is pushed.

On the write-side, the **TCSETA**, **TCSETAF**, **TCSETAW**, **TCGETA**, **TCSETS**, **TCSETSW**, **TCSETSF**, **TCGETS**, **TCSBRK**, **JWINSIZE**, **TIOCGWINSZ**, and **TIOCSWINSZ termio ioctl**(2) messages are processed and acknowledged. A hang up (such as stty 0) is converted to a zero length **M_DATA** message and passed downstream. **termio cflags** and window row and column information are stored locally one per stream. **M_DELAY** messages are discarded. All other messages are passed downstream unmodified.

On the read-side all messages are passed upstream unmodified with the following exceptions. All **M_READ** and **M_DELAY** messages are freed in both directions. An **ioctl TCSBRK** is converted to an **M_BREAK** message and passed upstream and an acknowledgement is returned downstream. An **ioctl TIOCSIGNAL** is converted into an **M_PCSIG** message, and passed upstream and an acknowledgement is returned downstream. Finally an **ioctl TIOCREMOTE** is converted into an **M_CTL** message, acknowledged, and passed upstream.

SEE ALSO

crash(1M), **ioctl**(2), **ldterm**(7), **pckt**(7), **pseudo**(1), **pty**(7), **streamio**(7), **stty**(1), **termio**(7)

NAME

pty – STREAMS pseudo-terminal driver

DESCRIPTION

The pseudo-terminal subsystem (**pty**) supports a pair of STREAMS-based devices called the master device and the slave device. The slave device provides processes with an interface that is identical to the terminal interface. However, whereas all devices that provide the terminal interface have some kind of hardware device behind them, the slave device has another process manipulating it through the master half of the pseudo terminal. Anything written on the master device is given to the slave as input and anything written on the slave device is presented as input on the master side.

The master device, called **ptm**, is accessed through the clone driver and is the controlling part of the system. The slave device, called **pts**, works with a line discipline module such as **ldterm**(7), a hardware emulation module such as **ptem**(7), and optionally with an **ioctl** translation module such as **ttcompat**(7) to provide a terminal interface to the user process. An optional packetizing module called **pckt**(7) is also provided to support "packet mode" when it is pushed on the master side.

The master device is opened via the **open**(2) system call with **/dev/ptmx** as the device to be opened. The clone open finds the next available minor device for that major device; a master device is available only if it and its corresponding slave device are not already open.

When the master device is opened, the corresponding slave device is automatically locked out, and no user may open that device until it is unlocked. A user may invoke **grantpt**(3C) to change the owner and permissions of the slave device to that of the user who is running the process. Once the permissions have been changed, the device may be unlocked by the user. Only the owner or a privileged user can access the slave device. The user then invokes **unlockpt**(3C) to unlock the slave device. The user calls **ptsname**(3C) to get the name of the slave device, and then invokes the **open** system call with the name that was returned by the function.

After both the master and slave devices have been opened, the user has two file descriptors that provide full-duplex communication using two streams. The two streams are automatically connected. The user may then push modules onto either side of the stream. The user also must push the **ptem** and **ldterm** modules onto the slave side to get terminal semantics.

The master and slave devices pass all STREAMS messages to their adjacent queues. Only **M_FLUSH** needs some processing. Because the read queue of one side is connected to the write queue of the other, the **FLUSHR** flag is changed to **FLUSHW** and vice versa.

When the master device is closed, an **M_HANGUP** message is sent to the slave device, which renders the device unusable. The process on the slave side gets the **errno** **ENXIO** when attempting to write on that stream, but it can read any data remaining on the stream head read queue. When all the data have been read, **read** returns 0, indicating that the stream can no longer be used.

On the last close of the slave device, a zero-length message is sent to the master side. When the master side application issues a read and 0 is returned, the user decides whether to issue a close, which dismantles the pseudo-terminal, or not close the master device so that the pseudo-tty subsystem will be available for another user to open the slave device.

ioctls

The master device supports the **ISPTM** and **UNLKPT** **ioctl**s that are used by the **grantpt**, **unlockpt**, and **ptsname** functions.

> The **ioctl ISPTM** determines whether the file descriptor is that of an open master device. On success, it returns the major/minor number (type **dev_t**) of the master device, which can be used to determine the name of the corresponding slave device. On failure it returns –1.
>
> The **ioctl UNLKPT** unlocks the master and slave devices. It returns 0 on success. On failure, it returns –1 and sets **errno** to **EINVAL**, indicating that the master device is not open.

The format of these commands is:

```
    int ioctl (int fd, int command, int arg);
```

where *command* is either **ISPTM** or **UNLKPT** and *arg* is 0.

The master side application is responsible for detecting an interrupt character and sending an interrupt signal **SIGINT** to the process on the slave side. This can be done as follows:

```
    ioctl (fd, TIOCSIGNAL, SIGINT);
```

where **SIGINT** is defined in the header file **signal.h**.

FILES

/dev/ptmx	pseudo-terminal master device
/dev/pts/*	pseudo-terminal slave devices

SEE ALSO

grantpt(3C), **ldterm**(7), **pckt**(7), **pseudo**(1), **ptem**(7), **ptsname**(3C), **ttcompat**(7), **unlockpt**(3C)

NAME

`rtc` – real time clock interface

DESCRIPTION

The `rtc` driver supports the real time clock chip, allowing it to be set with the correct local time and allowing the time to be read from the chip.

Ioctl Calls

`RTCRTIME`

This call is used to read the local time from the real time clock chip. The argument to the *ioctl* is the address of a buffer of **RTCNREG** unsigned characters (**RTCNREG** is defined in **sys/rtc.h**). The **ioctl** will fill in the buffer with the contents of the chip registers. Currently, **RTCNREG** is 14, and the meanings of the byte registers are as follows:

Register	Contents
0	Seconds
1	Second alarm
2	Minutes
3	Minute alarm
4	Hours
5	Hour alarm
6	Day of week
7	Date of month
8	Month
9	Year
A	Status register A
B	Status register B
C	Status register C
D	Status register D

For further information on the functions of these registers, see your hardware technical reference manual.

`RTCSTIME`

This call is used to set the time into the real time clock chip. The argument to **ioctl** is the address of a buffer of **RTCNREGP** unsigned characters (**RTCNREGP** is defined in **sys/rtc.h**). These bytes should be the desired chip register contents. Currently, **RTCNREGP** is 10, representing registers 0–9 as shown above. Note that only the super-user may open the real time clock device for writing and that the **RTCSTIME ioctl** will fail for any other than the super-user.

FILES

`/dev/rtc`

NAME

sad – STREAMS Administrative Driver

SYNOPSIS

```
#include <sys/types.h>
#include <sys/conf.h>
#include <sys/sad.h>
#include <sys/stropts.h>

int ioctl (int fildes, int command, . . . /* arg */);
```

DESCRIPTION

The STREAMS Administrative Driver provides an interface for applications to perform administrative operations on STREAMS modules and drivers. The interface is provided through **ioctl**(2) commands. Privileged operations may access the **sad** driver via **/dev/sad/admin**. Unprivileged operations may access the **sad** driver via **/dev/sad/user**.

fildes is an open file descriptor that refers to the **sad** driver. *command* determines the control function to be performed as described below. *arg* represents additional information that is needed by this command. The type of *arg* depends upon the command, but it is generally an integer or a pointer to a *command*-specific data structure.

Commands

The autopush facility [see **autopush**(1M)] allows one to configure a list of modules to be automatically pushed on a stream when a driver is first opened. Autopush is controlled by the next commands.

SAD_SAP Allows the administrator to configure the autopush information for the given device. *arg* points to a **strapush** structure which contains the following members:

```
uint    sap_cmd;
long    sap_major;
long    sap_minor;
long    sap_lastminor;
long    sap_npush;
uint    sap_list[MAXAPUSH] [FMNAMESZ + 1];
```

The **sap_cmd** field indicates the type of configuration being done. It may take on one of the following values:

SAP_ONE	Configure one minor device of a driver.
SAP_RANGE	Configure a range of minor devices of a driver.
SAP_ALL	Configure all minor devices of a driver.
SAP_CLEAR	Undo configuration information for a driver.

The **sap_major** field is the major device number of the device to be configured. The **sap_minor** field is the minor device number of the device to be configured. The **sap_lastminor** field is used only with the **SAP_RANGE** command, with which a range of minor devices between **sap_minor** and **sap_lastminor**, inclusive, are to be configured. The minor fields have no meaning for the **SAP_ALL** command. The **sap_npush** field indicates the number of modules to be automatically pushed when the device is opened. It must be less than or equal to **MAXAPUSH**, defined in **sad.h**. It must also be less than or equal to **NSTRPUSH**, the maximum number of modules that can be pushed on a stream, defined in the kernel master file. The field **sap_list** is an array of module names to be pushed in the order in which they appear in the list.

When using the **SAP_CLEAR** command, the user sets only **sap_major** and **sap_minor**. This will undo the configuration information for any of the other commands. If a previous entry was configured as **SAP_ALL**, **sap_minor** should be set to zero. If a previous entry was configured as **SAP_RANGE**, **sap_minor** should be set to the lowest minor device number in the range configured.

On failure, **errno** is set to the following value:

EFAULT	*arg* points outside the allocated address space.
EINVAL	The major device number is invalid, the number of modules is invalid, or the list of module names is invalid.
ENOSTR	The major device number does not represent a STREAMS driver.
EEXIST	The major-minor device pair is already configured.
ERANGE	The command is **SAP_RANGE** and **sap_lastminor** is not greater than **sap_minor**, or the command is **SAP_CLEAR** and **sap_minor** is not equal to the first minor in the range.
ENODEV	The command is **SAP_CLEAR** and the device is not configured for autopush.
ENOSR	An internal autopush data structure cannot be allocated.

SAD_GAP Allows any user to query the **sad** driver to get the autopush configuration information for a given device. *arg* points to a **strapush** structure as described in the previous command.

The user should set the **sap_major** and **sap_minor** fields of the **strapush** structure to the major and minor device numbers, respectively, of the device in question. On return, the **strapush** structure will be filled in with the entire information used to configure the device. Unused entries in the module list will be zero-filled.

On failure, **errno** is set to one of the following values:

EFAULT	*arg* points outside the allocated address space.
EINVAL	The major device number is invalid.

ENOSTR	The major device number does not represent a STREAMS driver.
ENODEV	The device is not configured for autopush.

SAD_VML Allows any user to validate a list of modules (for example, to see if they are installed on the system). *arg* is a pointer to a **str_list** structure with the following members:

```
int                 sl_nmods;
struct str_mlist    *sl_modlist;
```

The **str_mlist** structure has the following member:

```
char     l_name[FMNAMESZ+1];
```

sl_nmods indicates the number of entries the user has allocated in the array and **sl_modlist** points to the array of module names. The return value is 0 if the list is valid, 1 if the list contains an invalid module name, or -1 on failure. On failure, **errno** is set to one of the following values:

EFAULT	*arg* points outside the allocated address space.
EINVAL	The **sl_nmods** field of the **str_list** structure is less than or equal to zero.

SEE ALSO

intro(2), **ioctl**(2), **open**(2)

DIAGNOSTICS

Unless specified otherwise above, the return value from **ioctl** is 0 upon success and -1 upon failure with **errno** set as indicated.

NAME

`sc01` – CD-ROM Target Driver

DESCRIPTION

`sc01` is a Portable Device Interface (PDI)–compliant CD-ROM target driver that provides access to one or more CD-ROM drives. Each drive must be attached to a SCSI bus controlled by an PDI-compliant host adapter driver [for example, see **`adsc`**(7)].

Access to a particular drive is accomplished using the **`sc01`** device files located in **`/dev/[r]cdrom`**. Each device file identifies a particular drive based on the SCSI ID assigned to that drive. The binding between a device file and a CD-ROM drive is as follows:

`/dev/rcdrom/cd0`	CD-ROM drive with lowest SCSI ID
`/dev/rcdrom/cd1`	CD-ROM drive with next to lowest SCSI ID

and so on.

Most CD-ROM drives can handle CD-ROM disks that contain two types of data: informational data and audio data. The **`sc01`** driver allows access to both types of data.

A CD-ROM disk that contains informational data is treated as a random-access storage device, such as a hard disk. The information on the disk is divided into consecutively numbered, fixed-size (usually 2 Kbytes) sectors that can be accessed in any order. The standard tools for reading data from a random-access device, such as **`dd(1)`** or **`read(2)`**, can be used to read informational data from a CD-ROM. However, all write operations are prohibited.

Audio commands control the operation of the drive's audio output hardware (usually a headphone jack located on the drive). For example, the **`C_PLAYAUDIO`** ioctl causes the audio hardware to decode and convert the audio data to analog at a specific location on the disk, and play the audio on the drive's audio output hardware. Audio data is not returned to the host system.

All audio data commands are executed through the ioctl interface and often require a parameter structure that identifies the audio data to be acted upon. Unlike informational data, audio data is not referenced by a sector address. The methods used to identify a particular section of audio data should be described in the SCSI interface section of the reference manual supplied with your CD-ROM drive. Audio data cannot be read as if it were informational data, and informational data cannot be played using the drive's audio hardware.

ioctl Calls

The **`sc01`** driver uses several **`ioctl`**(2) commands, listed below. Many of these **`ioctl`**(2) commands provide a convenient method for sending one of the preselected SCSI commands directly to the drive. SCSI commands not explicitly supported by the **`sc01`** driver can be sent to the drive using the pass-through facility provided by the SDI host adapter driver. For an example, see **`adsc`**(7).

The following ioctls are used to identify a target driver and to get a pass-through major and minor number for a target device.

B_GETTYPE
: Returns the type of peripheral bus (for example, **scsi**) used and the name of this driver (for example, **sc01**).

B_GETDEV
: Returns the major and minor number of the pass-through device for the CD-ROM drive. For example, see **adsc**(7) for details.

The following ioctls cause the appropriate Group-0, Group-1, or Group-6 SCSI commands to be sent to the device. These commands are defined by the SCSI bus specification and should also be described in the SCSI Interface section of the reference manual supplied with your CD-ROM drive.

Group 0

C_TESTUNIT
: Sends a **Test Unit Ready** command to the device.

C_REZERO
: Sends a **Rezero Device** command to the device.

C_SEEK
: Sends a **Seek** command to the device.

C_INQUIR
: Sends an **Inquiry** command to the device and returns the resulting data back to the caller.

C_STARTUNIT
: Sends a **Start Unit** command to the device.

C_STOPUNIT
: Sends a **Stop Unit** command to the device.

C_PREVMV
: Sends a **Prevent Media Removal** command to the device.

C_ALLOMV
: Sends an **Allow Media Removal** command to the device.

Group-1

C_READCAPA
: Sends a **Read Capacity** command to the device and returns the data sent by the drive.

Group-6

C_AUDIOSEARCH
: Sends an **Audio Search** command to the device.

C_PLAYAUDIO
: Sends a **Play Audio** command to the device.

C_STILL
: Sends a **Still** command to the device.

`C_TRAYOPEN`
: Sends a `Tray Open` command to the device.

`C_TRAYCLOSE`
: Sends a `Tray Close` command to the device.

Note: The Group 6 IOCTL's support only the drives that are software compatible with the Toshiba XM-3201B.

The following ioctls are also supported by the `sc01` driver.

`B_GET_SUBDEVS`
: Returns the number of sub-devices supported by this driver (for example, 1).

Files

`/usr/include/sys/cd_ioctl.h`
`/usr/include/sys/cdrom.h`
`/usr/include/sys/sc01.h`
`/etc/conf/pack.d/sc01/space.c`
`/dev/[r]cdrom/cd*`
`/usr/include/sys/scsi.h`
`/usr/include/sys/sdi.h`
`/usr/include/sys/sdi_edt.h`

REFERENCES

`adsc`(7), `dpt`(7), `ioctl`(2), `mcis`(7), `sd01`(7), `st01`(7), `sw01`(7), `wd7000`(7)

NAME

sd01 – PDI disk target driver

DESCRIPTION

The **sd01** disk target driver is the device-level driver for both Small Computer System Interface (SCSI) hard disks, SCSI optical disks, and ESDI/ST506/IDE/MFM integral disks. It provides block and character (raw) access to the disk, and I/O controls (**ioctl**) to the disk. **sd01** sets up two levels of organization to the disk, to allow the disk to be shared with other operating systems, and provide efficient sized portions within the UNIX system.

The first level of organization of the disk by **sd01** is the partition table. The partition table divides the disk into pieces (called partitions) which serve as a logical disks. There are a maximum of 4 partitions for each disk. A partition has four characteristics: a start sector, a length, an operating system type (for example, UNIX, DOS, Extended DOS, and so on), and an active flag (which indicates the current bootable partition). A valid partition has at least the first three fields defined. A bootable valid partition has all four fields defined/on.

The partition table is maintained by the **fdisk**(1M) command. The **sd01** target driver searches the partition table for UNIX partitions. The active flag is used not only to indicate that a partition on the boot disk is bootable, but also indicates whether it is accessible (for example, a UNIX partition on the second disk which isn't active cannot be accessed).

Within a UNIX partition is the second level of organization of the disk. The UNIX partition is broken into contiguous sections called slices. The slices of a UNIX partition are defined by the Virtual Table Of Contents (VTOC). The VTOC provides the means to break up the UNIX partition in smaller pieces to better manage the space, to differentiate slices for special purposes, and to allow protection of some of the slices. The VTOC allows for a maximum of 16 slices per disk. A slice also has four characteristics: a start sector, a length, a slice type (for example, root, user, swap, stand, and so on), and permissions (valid and mountable/unmountable). A slice can contain a filesystem (for example, VXFS, S5, BFS, and so on), can be used as swap space for paging, or left to be organized by an application such as a database.

Several of the slices have required definitions as follows:

Slice 0	The whole UNIX partition; that is, it has the same start and length as the UNIX partition.
Slice 7	The boot slice which contains UNIX boot code (if it is the boot disk), the VTOC information, and the PDINFO (described later). This slice occupies sector 1 through 34 of UNIX partition.
Slice 8	The alternates slice, containing the table of remapped sectors, sectors which have been remapped, and spare sectors available for remapping.
Slice 9	Used in 4.0 and earlier UNIX releases additionally as the alternate track area. The alternates mechanism was consolidated in SVR4.2 to use one slice.

On the boot disk, there are several other slices which also have required definitions:

Slice 1 The root filesystem

Slice 2 The swap slice

Slice 10
The boot slice which contains the BFS filesystem

Finally, on a boot disk, the optional filesystems are organized as follows:

Slice 3 The `/usr` filesystem

Slice 4 The `/home` filesystem

Slice 5 Points to the first DOS partition, if defined

Slice 6 The dump slice (holds memory dumps)

Slice 11
The `/var` filesystem

Slice 12
The `/home2` filesystem

Slice 13
The `/tmp` filesystem

Slice 14
Points to the second DOS partition, if defined

Slice 15
Points to the third DOS partition, if defined

The slices of a disk are represented by device nodes, which have the major number for **sd01** and a minor number pointing to one of slices. Since there are sixteen slices, there are therefore sixteen minors per disk, so minor 0 through 15 are for the first disk, 16 through 31 are for the second disk, 32 through 47 for the third disk and so on. The system supports 256 minors per major number so thus there are 16 sets of disk devices per major number. **sd01** supports multiple major numbers, and currently supports 7 major numbers, which allows for up to 112 disks. When the disk device is opened, the partition table and the VTOC are read by **sd01** to fill out its tables of partitions and slices.

Mapping of bad blocks is performed dynamically and automatically by the **sd01** disk driver, without user intervention and without retaining a fixed bad block log on the disk. The SCSI direct-access controllers reassign the defective blocks to an area on the disk reserved for this purpose. The **sd01** disk driver can map both marginal bad blocks (that is, readable with some difficulty) and actual bad blocks (that is, unreadable). The **sd01** driver does not map or report a bad block residing in a non-UNIX System (that is, MS-DOS) partition of the disk. In addition, even with dynamic bad block handling, it is possible for damage to occur that cannot be mapped automatically. This means that you may have to restore the file system from the last full backup, if the bad block occurs in a critical area of the disk which cannot tolerate bad sectors.

The **sd01** disk driver reports problems with driver error messages. The error numbers in the error messages identify the type of error. For SCSI sense codes, extended sense codes, and command codes, see the file **/usr/include/sys/scsi.h**. For SDI return codes, see the file

/usr/include/sys/sdi.h.

The **sd01** driver receives command requests from the kernel through the Input/Output (I/O) control call **ioctl**(2). The **sd01** driver generates the requested commands and passes them to the host adapter driver. When command execution is complete, the host adapter driver notifies the **sd01** driver through an interrupt. After this notification, the **sd01** driver performs any required error recovery and indicates to the kernel that the I/O request is complete.

The files in the **/dev/dsk** directory access the disk through the system's normal buffering mechanism, and may be read and written without regard to physical disk records.

There is also a [**r**] raw interface that provides for direct transmission between the disk and the user's read or write buffer. A single read or write call results in exactly one I/O operation. Therefore, raw I/O is considerably more efficient when many words are transmitted. The names of the raw disk files contain **/dev/rdsk** and have the same form as the **/dev/dsk** files.

In raw I/O, the buffer must begin on a 512-byte boundary, and transfer counts must be integral multiples of 512 bytes.

The special device file names associated with the **sd01** disk driver have the form:

> **/dev/**[**r**]**dsk/c#t#d#s#**

The naming convention for the **sd01** disk special device file name components is as follows:

[**r**]	The optional **r** in **/[r]dsk** denotes a raw (that is, character) device; **/dsk** without the optional **r** indicates a block device
c#	# is the occurrence of the host adapter board in the system (that is, card number), ranging are from 0-2 (machine dependent)
t#	# is the target controller number, ranging are from 0-F hexadecimal
d#	# is the logical unit number of the disk device, ranging are from 0-3, since each target controller currently supports up to four disks
s#	# is the disk slice number or VTOC partition number, ranging are from 0-F hexadecimal

The disk parameters—number of cylinders, heads, and sectors per track—are obtained at driver initialization (init) time. If the disk is an ESDI/ST506/IDE drive, the CMOS contains parameters for the first two disks. For SCSI disks, a read capacity command is issued, and the disk parameters are calculated based on a disk geometry of 32 sectors per track and 64 heads. This geometry is for drives with 512 byte sectors. It is adjusted if the number of bytes per sector changes. The disk parameters are stored in Physical Descriptor Information (PDINFO) structure which is stored in the boot slice of the disk. The PDINFO is kept as a sanity check against those found at driver init time.

When the machine is booted, the primary boot code (BIOS) looks in the **fdisk** table for the active partition and jumps to sector 0 of that partition to find the first-stage bootstrap. If the first-stage bootstrap is over one sector in length, it is the responsibility of the first-stage bootstrap to understand this. The boot code will read in the VTOC to locate the BFS filesystem. It will then load the kernel and begin executing the kernel.

ioctl Calls

The **ioctl** calls used by the **sd01** driver to control the reading and writing of data to disk are as follows:

V_CONFIG

Used to modify the parameters (cylinders and heads) of a disk device. Its usage is not recommended and it is no longer used in any of the system commands. The argument to the **ioctl** is the address of one of the following structures, defined in **sys/vtoc.h**, containing the new configuration parameters:

```
union io_arg {
struct {
ushortncyl;/* Number of cylinders */
uncharnhead;/* Heads/cylinder */
uncharnsec;/* Sectors/track */
ushortsecsiz;/* Bytes/sector */
} ia_cd;
}
```

Note that it is not possible to change the sector size on the hard disk with this **ioctl**, and that an attempt to do so results in the **ioctl** failing, with **errno** set to **EINVAL**. This call is provided for backward compatibility with any commands which use it. This call should no longer be used and will be removed in the future.

V_REMOUNT

Forces **sd01** to re-read the VTOC on the next open operation of the drive. It fails if any slice other than slice 0 is currently open, since the VTOC information cannot be updated while a process is using a slice. This is used by **disksetup** when it changes the VTOC to signal **sd01** to update its internal tables.

V_GETPARMS

Gets information about the current drive configuration. The argument to the **ioctl** is the address of the following structure, defined in **sys/vtoc.h**, which are filled in by the **ioctl**:

```
struct disk_parms {
chardp_type;/* Disk type (see below) */
unchardp_heads;/* No. of heads */
ushortdp_cyls;/* No. of cylinders */
unchardp_sectors;/* No. of sectors/track */
ushortdp_secsiz;/* No. of bytes/sector */
ushortdp_ptag;/* Currently not used */
ushortdp_pflag;/* Currently not used */
daddr_tdp_pstartsec;/* Starting abs. sector no. */
daddr_tdp_pnumsec;/* Currently not used */
}

/* Disk types */
```

```
#defineDPT_NOTDISK0/* Not a disk device */
#defineDPT_WINI1/* Winchester disk */
#defineDPT_FLOPPY2/* Floppy */
#defineDPT_OTHER3/* Other type of disk */
#defineDPT_SCSI_HD4/* SCSI hard disk */
#defineDPT_SCSI_OD5/* SCSI optical disk */

/* Partition tag */
#defineV_BOOT1/* Bootable partition */
#defineV_ROOT2/* Root filesystem */
#defineV_SWAP3/* Swap slice */
#defineV_USR4/* User filesystem */
#defineV_BACKUP5/* Entire disk */
#defineV_ALTS6/* Alternate sectors (SVR4.0 and earlier)  */
#defineV_OTHER7/* Non-UNIX System partition */
#defineV_ALTTRK8/* Alternate tracks (SVR4.0 and earlier) */
#defineV_STAND9/* stand (BFS) filesystem */
#defineV_VAR0A/* Var filesystem */
#defineV_HOME0B/* Home filesystem */
#defineV_DUMP0C/* Dump slice */
#defineV_ALTSCTR0D/* Alternate sectors (SVR4.2)  */

/* Partition flag */
#defineV_UNMNT0x001/* Unmountable partition */
#defineV_RONLY0x010/* Read only partition */
#defineV_OPEN0x100/* Partition open */
#defineV_VALID0x200/* Partition valid to use */
```

For SCSI disks the disk type is **DPT_SCSI_HD**. For ESDI/ST506/IDE disks the disk type is **DPT_WINI**.

V_PDLOC

Returns the logical sector address of the **pdinfo** structure. The value is returned in **pdloc**.

```
unsignedlongpdloc;
```

V_RDABS/V_WRABS

Used as a means for reading/writing any sector on the hard disk. Only users with root privilege can freely access any sector. Users who do not have root privilege can access the partition table (sector 0) or the boot slice (to allow access to the VTOC). The absolute sector address to be written to is placed in **abs_sec**. The data for the sector is read to or written from **abs_buf**. The size of **abs_buf** should be *disk_parms.dp_secsize* for the current drive. Note that both the first cylinder (containing the **fdisk** table, first-stage bootstrap and VTOC) and the first track of the active partition (containing the first-stage bootstrap) can only be accessed using partition 0, since these tracks are normally not considered part of any other partition in the VTOC. The **absio** structure is defined in **sys/vtoc.h**.

```
struct absio {
daddr_tabs_sec;/* Absolute sector no. (from 0) */
char*abs_buf;/* Sector buffer */
};
```

V_PREAD/V_PWRITE
Used to read or write any size data block on the disk, regardless of the physical sector size. Only users with root privilege can use these calls. The starting logical sector address to be written to or read from is placed in **sectst**, the number of bytes to be transferred is placed in **datasz**, the data to be transferred is placed in **memaddr**, and the **phyio** structure is defined in **sys/vtoc.h**.

```
struct phyio {
intretval;/* Return value */
unsignedlong sectst;/* Sector address */
unsignedlong memaddr;/* Buffer address */
unsignedlong datasz;/* Transfer size in bytes */
};
```

V_PDREAD/V_PDWRITE
Used to read or write the Physical Description sector on the disk, regardless of this sector's location. Only users with root privilege can use these calls. The starting logical sector address to be written to or read from is assigned by the **sd01** driver, the physical sector size of the disk must be placed in **datasz**, the data to be transferred is placed in **memaddr**, and the **phyio** structure is defined in **sys/vtoc.h**.

SD_PDLOC
Returns the physical sector address of the **pdinfo** structure. The value is returned in **pdloc**.

```
unsignedlongpdloc;
```

SDI_RESERVE
Reserves a SCSI disk for a processor.

SDI_RELEASE
Releases a SCSI disk from a processor.

SDI_RESTAT
Returns device reservation status.

The following **ioctl** commands are used to identify a target driver and to get pass-through major and minor numbers for a target device:

B_GETTYPE
Returns the bus name (for example, **scsi**) and device driver name (for example, **sd01**) of a specific device.

B_GETDEV
Returns the pass-through major and minor numbers to the calling utility, allowing creation of a pass-through special device file.

Files

```
/dev/dsk/*
/dev/rdsk/*
/usr/include/sys/scsi.h
/usr/include/sys/sdi.h
/usr/include/sys/sdi_edt.h
/usr/include/sys/vtoc.h
```

NOTES

The `sd01` driver retries failed transfers up to two times depending on the error type. Certain errors are not retried. `sd01` displays an appropriate message upon encountering an error during the transfer.

The VTOC and second-stage bootstrap require that no bad sectors occur in the first 30 sectors of the UNIX System partition on the disk. When a marginal bad block occurs, the driver's warning indicates that the controller's error-correction algorithm successfully recovered from an error. This may be a symptom of a sector going bad.

REFERENCES

adsc(7), **disksetup**(1M), **dpt**(7), **edvtoc**(1M), **fdisk**(1M), **fs**(4), **ioctl**(2) **mcis**(7) **mount**(1M), **prtvtoc**(1M), **sc01**(7), **st01**(7), **sw01**(7), **wd7000**(7)

NAME

`sockio` – `ioctl`s that operate directly on sockets

SYNOPSIS

`#include <sys/sockio.h>`

DESCRIPTION

The `ioctl`s listed in this manual page apply directly to sockets, independent of any underlying protocol. The **`setsockopt`** call (see **`getsockopt`**(3N)) is the primary method for operating on sockets, rather than on the underlying protocol or network interface. `ioctl`s for a specific network interface or protocol are documented in the manual page for that interface or protocol.

`SIOCSPGRP`	The argument is a pointer to an **`int`**. Set the process-group ID that will subsequently receive **`SIGIO`** or **`SIGURG`** signals for the socket referred to by the descriptor passed to **`ioctl`** to the value of that **`int`**.
`SIOCGPGRP`	The argument is a pointer to an **`int`**. Set the value of that **`int`** to the process-group ID that is receiving **`SIGIO`** or **`SIGURG`** signals for the socket referred to by the descriptor passed to **`ioctl`**.
`SIOCCATMARK`	The argument is a pointer to an **`int`**. Set the value of that **`int`** to 1 if the read pointer for the socket referred to by the descriptor passed to **`ioctl`** points to a mark in the data stream for an out-of-band message. Set the value of that **`int`** to 0 if the read pointer for the socket referred to by the descriptor passed to **`ioctl`** does not point to a mark in the data stream for an out-of-band message.

SEE ALSO

`ioctl`(2) **`getsockopt`**(3N),

NAME

st01 – Portable Device Interface (PDI) tape target driver

DESCRIPTION

The **st01** tape driver receives command requests from the kernel through the **read**(2), **write**(2), and **ioctl**(2) system calls. The **st01** driver generates the appropriate commands and passes them through the host adapter driver to the tape device. When command execution is complete, the host adapter driver notifies **st01** through an interrupt. After this notification, **st01** performs any required error recovery, and indicates to the kernel that the Input/Output (I/O) request is complete. The **st01** driver operates independently of the hardware used to communicate with the HBA bus.

I/O requests must be in length a multiple of the tape block length. The default value is 512 bytes.

Only raw character interface files are provided. When opened, the tape is assumed to be positioned as desired. If a retension-on-open special file is opened, the tape is retensioned before any I/O is performed. When a **T_RWD**, **T_RETENSION**, **T_LOAD**, or **T_UNLOAD** **ioctl** is requested and the tape has been written, two file marks are written before the **ioctl** is executed.

The **open**(2) on a tape device can fail is a tape is not inserted, resulting in the error report **EIO**. An **open**(2) can also fail if the tape controller associated with the special file is not detected by the driver. In this case, the error reported is **ENXIO**.

The following table lists the actions that occur on **close**. depending on whether the file is designated as rewind or no-rewind, and if the tape was written or read:

Rewind on Close?	Tape Read?	Tape Written?	Action on Close
Yes	Yes	N/A	Rewind tape.
Yes	N/A	Yes	Write two file marks and rewind tape.
Yes	No	No	Rewind tape
No	No	No	No tape movement
No	Yes	N/A	Position tape after next file mark
No	N/A	Yes	Write one file mark and position tape after this file mark

A read occurring when the tape is positioned immediately before a file mark returns zero(0) bytes, and the tape is positioned after the file mark. As with other raw devices, seeks are ignored. Some tape devices allow both reads and writes to occur between rewinds; the **st01** driver supports these devices.

ioctl Calls

The following **ioctl** calls are used by the **st01** driver to control tape positioning:

T_SFF/T_SFB Positions the tape forward or backward *arg* [see **ioctl**(2)] number of file marks from the current tape head position toward the End-of-Tape (EOT) or Beginning-of-Tape (BOT). Forward movement of the tape leaves the tape positioned on the EOT side of a file mark or at EOT, and backwards movement leaves the tape positioned on the BOT side of a file mark or at BOT. A backward positioning operation causes the next read to return 0 bytes unless *arg* is greater than the number of file marks between the current position and BOT. The value of *arg* must be a positive integer. A value of 0 is not considered an error, but does not result in any tape movement.

T_SBF/T_SBB Positions the tape forward or backward *arg* number of blocks from the current tape head position toward the EOT or BOT. Upon command completion, the tape head is positioned in the gap between tape blocks. Thus, skipping a block forward advances to the next block, and skipping a block backward retreats to the last block. The value of *arg* must be a positive integer. Upon any attempt to skip over a file mark, the tape is positioned on the EOT/BOT side of the file mark for forward/backward movement, and the positioning operation ceases. A value of 0 is not considered an error, but does not result in any tape movement.

T_RWD Rewinds the tape from the current tape position to the BOT. Two file marks are written before the rewind if the tape has been written. This command does not unload the tape.

T_WRFILEM Writes file marks to the tape. The value of *arg* defines the number of consecutive file marks to be written. If an error occurs while writing file marks, the number of file marks that have been successfully written is indeterminate.

T_EOD Positions the tape just beyond the last file mark.

T_STD Defines the recording density of the tape media being used. The numeric density code used is as defined in the SCSI-2 draft specification.

T_PREVMV Locks the tape in the drive. This prevention may be in the form of a mechanical lock or an LED to indicate the device is in use. **T_PREVMV** is supported only on devices that implement this feature. For example, ICT devices are among those which do not support this ioctl.

T_ALLOMV Unlocks the tape in the drive. This command is used to undo the lock created by **T_PREVMV**. **T_ALLOMV** is supported only on devices that implement this feature. For example, ICT devices are among those which do not support this ioctl.

T_LOAD Loads the tape media and position the tape BOT.

T_UNLOAD Unloads the tape. Most devices rewind the tape before unloading. Devices capable of ejecting the tape will do so in response to this command.

T_ERASE Erases the tape, from BOT to EOT. If the tape is not positioned at BOT, the tape is positioned at BOT before performing the erase function.

`T_RDBLKLEN`	Returns the minimum and maximum block lengths supported by the tape device. The value of *arg* must be a **`struct blklen`**. See the file **`/usr/include/sys/st01.h`** for more information.
`T_WRBLKLEN`	Sets the current block length for the tape device. The value of *arg* must be a **`struct blklen`** with both *max_blen* and *min_blen* set to the desired block length. See the file **`/usr/include/sys/st01.h`** for more information.
`T_RETENSION`	Retensions the tape in the drive, running the tape at high speed from BOT to EOT, and then back again. The retension operation leaves the tape positioned at BOT.

The following **`ioctl`** commands identify a target driver and get a pass-through major and minor number for a target device.

`B_GETTYPE`	Gets the bus name (for example, **`scsi`**) and device driver name (for example, **`st01`**) of a specific device
`B_GETDEV`	Gets the pass-through major and minor number to the calling utility, allowing creation of a pass-through special device file.

Files

`/usr/include/sys/st01_ioctl.h`
`/usr/include/sys/st01.h`
`/usr/include/sys/sdi_edt.h`

NOTES

Once any drive error is encountered, the driver will not perform any other functions until the file is closed.

The **`st01`** tape driver does not always require block sizes that are in multiples of 512 bytes, but block size is device dependent. You should set the tape driver to use the block size supported by the tape device. Failure to set the block size correctly will result in an error when the driver attempts to write a block of the unsupported size.

The tape driver does not support the use of the **`sar`** command.

REFERENCES

`adsc`(7), **`close`**(2), **`dpt`**(7), **`ioctl`**(2), **`mcis`**(7), **`read`**(2), **`sc01`**(7), **`sd01`**(7), **`sw01`**(7), **`wd7000`**(7), **`write`**(2)

NAME

streamio – STREAMS **ioctl** commands

SYNOPSIS

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

int ioctl (int fildes, int command, ... /* arg */);
```

DESCRIPTION

STREAMS [see **intro**(2)] **ioctl** commands are a subset of the **ioctl**(2) system calls which perform a variety of control functions on streams.

fildes is an open file descriptor that refers to a stream. *command* determines the control function to be performed as described below. *arg* represents additional information that is needed by this command. The type of *arg* depends upon the command, but it is generally an integer or a pointer to a *command*-specific data structure. The *command* and *arg* are interpreted by the stream head. Certain combinations of these arguments may be passed to a module or driver in the stream.

Since these STREAMS commands are a subset of **ioctl**, they are subject to the errors described there. In addition to those errors, the call will fail with **errno** set to **EINVAL**, without processing a control function, if the stream referenced by *fildes* is linked below a multiplexor, or if *command* is not a valid value for a stream.

Also, as described in **ioctl**, STREAMS modules and drivers can detect errors. In this case, the module or driver sends an error message to the stream head containing an error value. This causes subsequent system calls to fail with **errno** set to this value.

Command Functions

The following **ioctl** commands, with error values indicated, are applicable to all STREAMS files:

I_PUSH Pushes the module whose name is pointed to by *arg* onto the top of the current stream, just below the stream head. If the stream is a pipe, the module will be inserted between the stream heads of both ends of the pipe. It then calls the open routine of the newly-pushed module. On failure, **errno** is set to one of the following values:

EINVAL	Invalid module name.
EFAULT	*arg* points outside the allocated address space.
ENXIO	Open routine of new module failed.
ENXIO	Hangup received on *fildes*.

I_POP Removes the module just below the stream head of the stream pointed to by *fildes*. To remove a module from a pipe requires that the module was pushed on the side it is being removed from. *arg* should be 0 in an **I_POP** request. On failure, **errno** is set to one of the following values:

EINVAL	No module present in the stream.

	ENXIO	Hangup received on *fildes*.

I_LOOK Retrieves the name of the module just below the stream head of the stream pointed to by *fildes*, and places it in a null terminated character string pointed at by *arg*. The buffer pointed to by *arg* should be at least **FMNAMESZ**+1 bytes long. A **#include <sys/conf.h>** declaration is required. On failure, **errno** is set to one of the following values:

EFAULT	*arg* points outside the allocated address space.
EINVAL	No module present in stream.

I_FLUSH This request flushes all input and/or output queues, depending on the value of *arg*. Valid *arg* values are:

FLUSHR	Flush read queues.
FLUSHW	Flush write queues.
FLUSHRW	Flush read and write queues.

If a pipe or FIFO does not have any modules pushed, the read queue of the stream head on either end is flushed depending on the value of *arg*.

If **FLUSHR** is set and *fildes* is a pipe, the read queue for that end of the pipe is flushed and the write queue for the other end is flushed. If *fildes* is a FIFO, both queues are flushed.

If **FLUSHW** is set and *fildes* is a pipe and the other end of the pipe exists, the read queue for the other end of the pipe is flushed and the write queue for this end is flushed. If *fildes* is a FIFO, both queues of the FIFO are flushed.

If **FLUSHRW** is set, all read queues are flushed, that is, the read queue for the FIFO and the read queue on both ends of the pipe are flushed.

Correct flush handling of a pipe or FIFO with modules pushed is achieved via the **pipemod** module. This module should be the first module pushed onto a pipe so that it is at the midpoint of the pipe itself.

On failure, **errno** is set to one of the following values:

ENOSR	Unable to allocate buffers for flush message due to insufficient STREAMS memory resources.
EINVAL	Invalid *arg* value.
ENXIO	Hangup received on *fildes*.

I_FLUSHBAND

Flushes a particular band of messages. *arg* points to a **bandinfo** structure that has the following members:

```
unsigned char   bi_pri;
int             bi_flag;
```

The **bi_flag** field may be one of **FLUSHR**, **FLUSHW**, or **FLUSHRW** as described earlier.

`I_SETSIG`	Informs the stream head that the user wants the kernel to issue the **`SIGPOLL`** signal [see **`signal`**(2)] when a particular event has occurred on the stream associated with *fildes*. **`I_SETSIG`** supports an asynchronous processing capability in STREAMS. The value of *arg* is a bitmask that specifies the events for which the user should be signaled. It is the bitwise-OR of any combination, except where noted, of the following constants:

`S_INPUT`	Any message other than an **`M_PCPROTO`** has arrived on a stream head read queue. This event is maintained for compatibility with prior releases. This is set even if the message is of zero length.
`S_RDNORM`	An ordinary (non-priority) message has arrived on a stream head read queue. This is set even if the message is of zero length.
`S_RDBAND`	A priority band message (band > 0) has arrived on a stream head read queue. This is set even if the message is of zero length.
`S_HIPRI`	A high priority message is present on the stream head read queue. This is set even if the message is of zero length.
`S_OUTPUT`	The write queue just below the stream head is no longer full. This notifies the user that there is room on the queue for sending (or writing) data downstream.
`S_WRNORM`	This event is the same as **`S_OUTPUT`**.
`S_WRBAND`	A priority band greater than 0 of a queue downstream exists and is writable. This notifies the user that there is room on the queue for sending (or writing) priority data downstream.
`S_MSG`	A STREAMS signal message that contains the **`SIGPOLL`** signal has reached the front of the stream head read queue.
`S_ERROR`	An **`M_ERROR`** message has reached the stream head.
`S_HANGUP`	An **`M_HANGUP`** message has reached the stream head.
`S_BANDURG`	When used in conjunction with **`S_RDBAND, SIGURG`** is generated instead of **`SIGPOLL`** when a priority message reaches the front of the stream head read queue.

A user process may choose to be signaled only of high priority messages by setting the *arg* bitmask to the value **`S_HIPRI`**.

Processes that want to receive **`SIGPOLL`** signals must explicitly register to receive them using **`I_SETSIG`**. If several processes register to receive this signal for the same event on the same stream, each process will be signaled when the event occurs.

If the value of *arg* is zero, the calling process will be unregistered and will not receive further **SIGPOLL** signals. On failure, **errno** is set to one of the following values:

EINVAL	*arg* value is invalid or *arg* is zero and process is not registered to receive the **SIGPOLL** signal.
EAGAIN	Allocation of a data structure to store the signal request failed.

I_GETSIG Returns the events for which the calling process is currently registered to be sent a **SIGPOLL** signal. The events are returned as a bitmask pointed to by *arg*, where the events are those specified in the description of **I_SETSIG** above. On failure, **errno** is set to one of the following values:

EINVAL	Process not registered to receive the **SIGPOLL** signal.
EFAULT	*arg* points outside the allocated address space.

I_FIND Compares the names of all modules currently present in the stream to the name pointed to by *arg*, and returns 1 if the named module is present in the stream. It returns 0 if the named module is not present. On failure, **errno** is set to one of the following values:

EFAULT	*arg* points outside the allocated address space.
EINVAL	*arg* does not contain a valid module name.

I_PEEK Allows a user to retrieve the information in the first message on the stream head read queue without taking the message off the queue. **I_PEEK** is analogous to **getmsg**(2) except that it does not remove the message from the queue. *arg* points to a **strpeek** structure which contains the following members:

```
struct strbuf    ctlbuf;
struct strbuf    databuf;
long             flags;
```

The **maxlen** field in the **ctlbuf** and **databuf** **strbuf** structures [see **getmsg**(2)] must be set to the number of bytes of control information and/or data information, respectively, to retrieve. **flags** may be set to **RS_HIPRI** or 0. If **RS_HIPRI** is set, **I_PEEK** will look for a high priority message on the stream head read queue. Otherwise, **I_PEEK** will look for the first message on the stream head read queue.

I_PEEK returns 1 if a message was retrieved, and returns 0 if no message was found on the stream head read queue. It does not wait for a message to arrive. On return, **ctlbuf** specifies information in the control buffer, **databuf** specifies information in the data buffer, and **flags** contains the value **RS_HIPRI** or 0. On failure, **errno** is set to the following value:

EFAULT	*arg* points, or the buffer area specified in **ctlbuf** or **databuf** is, outside the allocated address space.

	EBADMSG	Queued message to be read is not valid for **I_PEEK**
	EINVAL	Invalid value for **flags**.
I_SRDOPT	Sets the read mode [see **read**(2)] using the value of the argument *arg*. Valid *arg* values are:	
	RNORM	Byte-stream mode, the default.
	RMSGD	Message-discard mode.
	RMSGN	Message-nondiscard mode.

Setting both **RMSGD** and **RMSGN** is an error. **RMSGD** and **RMSGN** override **RNORM**.

In addition, treatment of control messages by the stream head may be changed by setting the following flags in *arg*:

RPROTNORM	Fail **read** with **EBADMSG** if a control message is at the front of the stream head read queue. This is the default behavior.
RPROTDAT	Deliver the control portion of a message as data when a user issues **read**.
RPROTDIS	Discard the control portion of a message, delivering any data portion, when a user issues a **read**.

On failure, **errno** is set to the following value:

EINVAL	*arg* is not one of the above valid values.
EINVAL	Both **RMSGD** and **RMSGN** are set.

I_GRDOPT Returns the current read mode setting in an **int** pointed to by the argument *arg*. Read modes are described in **read**(2). On failure, **errno** is set to the following value:

EFAULT	*arg* points outside the allocated address space.

I_NREAD Counts the number of data bytes in data blocks in the first message on the stream head read queue, and places this value in the location pointed to by *arg*. The return value for the command is the number of messages on the stream head read queue. For example, if zero is returned in *arg*, but the **ioctl** return value is greater than zero, this indicates that a zero-length message is next on the queue. On failure, **errno** is set to the following value:

EFAULT	*arg* points outside the allocated address space.

I_FDINSERT Creates a message from user specified buffer(s), adds information about another stream and sends the message downstream. The message contains a control part and an optional data part. The data and control parts to be sent are distinguished by placement in separate buffers, as described below.

arg points to a **strfdinsert** structure which contains the following members:

```
struct strbuf     ctlbuf;
struct strbuf     databuf;
long              flags;
int               fildes;
int               offset;
```

The **len** field in the **ctlbuf strbuf** structure [see **putmsg**(2)] must be set to the size of a pointer plus the number of bytes of control information to be sent with the message. *fildes* in the **strfdinsert** structure specifies the file descriptor of the other stream. **offset**, which must be word-aligned, specifies the number of bytes beyond the beginning of the control buffer where **I_FDINSERT** will store a pointer. This pointer will be the address of the read queue structure of the driver for the stream corresponding to **fildes** in the **strfd-insert** structure. The **len** field in the **databuf strbuf** structure must be set to the number of bytes of data information to be sent with the message or zero if no data part is to be sent.

flags specifies the type of message to be created. An ordinary (non-priority) message is created if **flags** is set to 0, a high priority message is created if **flags** is set to **RS_HIPRI**. For normal messages, **I_FDINSERT** will block if the stream write queue is full due to internal flow control conditions. For high priority messages, **I_FDINSERT** does not block on this condition. For normal messages, **I_FDINSERT** does not block when the write queue is full and **O_NDELAY** or **O_NONBLOCK** is set. Instead, it fails and sets **errno** to **EAGAIN**.

I_FDINSERT also blocks, unless prevented by lack of internal resources, waiting for the availability of message blocks, regardless of priority or whether **O_NDELAY** or **O_NONBLOCK** has been specified. No partial message is sent. On failure, **errno** is set to one of the following values:

EAGAIN	A non-priority message was specified, the **O_NDELAY** or **O_NONBLOCK** flag is set, and the stream write queue is full due to internal flow control conditions.
ENOSR	Buffers could not be allocated for the message that was to be created due to insufficient STREAMS memory resources.
EFAULT	*arg* points, or the buffer area specified in **ctlbuf** or **databuf** is, outside the allocated address space.
EINVAL	One of the following: **fildes** in the **strfdinsert** structure is not a valid, open stream file descriptor; the size of a pointer plus **offset** is greater than the **len** field for the buffer specified through **ctlptr**; **offset** does not specify a properly-aligned location in the data buffer; an undefined value is stored in **flags**.

ENXIO Hangup received on **fildes** of the **ioctl** call or **fildes** in the **strfdinsert** structure.

ERANGE The **len** field for the buffer specified through **databuf** does not fall within the range specified by the maximum and minimum packet sizes of the topmost stream module, or the **len** field for the buffer specified through **databuf** is larger than the maximum configured size of the data part of a message, or the **len** field for the buffer specified through **ctlbuf** is larger than the maximum configured size of the control part of a message.

I_FDINSERT can also fail if an error message was received by the stream head of the stream corresponding to **fildes** in the **strfd-insert** structure. In this case, **errno** will be set to the value in the message.

I_STR Constructs an internal STREAMS ioctl message from the data pointed to by *arg*, and sends that message downstream.

This mechanism is provided to send user **ioctl** requests to downstream modules and drivers. It allows information to be sent with the **ioctl**, and will return to the user any information sent upstream by the downstream recipient. **I_STR** blocks until the system responds with either a positive or negative acknowledgement message, or until the request "times out" after some period of time. If the request times out, it fails with **errno** set to **ETIME**.

At most, one **I_STR** can be active on a stream. Further **I_STR** calls will block until the active **I_STR** completes at the stream head. The default timeout interval for these requests is 15 seconds. The **O_NDELAY** and **O_NONBLOCK** [see **open**(2)] flags have no effect on this call.

To send requests downstream, *arg* must point to a **strioctl** structure which contains the following members:

```
int     ic_cmd;
int     ic_timout;
int     ic_len;
char    *ic_dp;
```

ic_cmd is the internal **ioctl** command intended for a downstream module or driver and **ic_timout** is the number of seconds (–1 = infinite, 0 = use default, >0 = as specified) an **I_STR** request will wait for acknowledgement before timing out. The default timeout is infinite. **ic_len** is the number of bytes in the data argument and **ic_dp** is a pointer to the data argument. The **ic_len** field has two uses: on input, it contains the length of the data argument passed in, and on return from the command, it contains the number of bytes being returned to the user (the buffer pointed to by **ic_dp** should be large enough to contain the maximum amount of data that any module or the driver in the stream can return).

The stream head will convert the information pointed to by the **strioctl** structure to an internal **ioctl** command message and send it downstream. On failure, **errno** is set to one of the following values:

ENOSR	Unable to allocate buffers for the **ioctl** message due to insufficient STREAMS memory resources.
EFAULT	*arg* points, or the buffer area specified by **ic_dp** and **ic_len** (separately for data sent and data returned) is, outside the allocated address space.
EINVAL	**ic_len** is less than 0 or **ic_len** is larger than the maximum configured size of the data part of a message or **ic_timout** is less than –1.
ENXIO	Hangup received on *fildes*.
ETIME	A downstream **ioctl** timed out before acknowledgement was received.

An **I_STR** can also fail while waiting for an acknowledgement if a message indicating an error or a hangup is received at the stream head. In addition, an error code can be returned in the positive or negative acknowledgement message, in the event the ioctl command sent downstream fails. For these cases, **I_STR** will fail with **errno** set to the value in the message.

I_SWROPT Sets the write mode using the value of the argument *arg*. Legal bit settings for *arg* are:

SNDZERO	Send a zero-length message downstream when a write of 0 bytes occurs.

To not send a zero-length message when a write of 0 bytes occurs, this bit must not be set in *arg*.

On failure, **errno** may be set to the following value:

EINVAL	*arg* is not the above valid value.

I_GWROPT Returns the current write mode setting, as described above, in the **int** that is pointed to by the argument *arg*.

I_SENDFD Requests the stream associated with *fildes* to send a message, containing a file pointer, to the stream head at the other end of a stream pipe. The file pointer corresponds to *arg*, which must be an open file descriptor.

I_SENDFD converts *arg* into the corresponding system file pointer. It allocates a message block and inserts the file pointer in the block. The user ID and group ID associated with the sending process are also inserted. This message is placed directly on the read queue [see **intro**(2)] of the stream head at the other end of the stream pipe to which it is connected. On failure, **errno** is set to one of the following values:

EAGAIN	The sending stream is unable to allocate a message block to contain the file pointer.
EAGAIN	The read queue of the receiving stream head is full and cannot accept the message sent by **I_SENDFD**.
EBADF	*arg* is not a valid, open file descriptor.
EINVAL	*fildes* is not connected to a stream pipe.
ENXIO	Hangup received on *fildes*.

I_RECVFD Retrieves the file descriptor associated with the message sent by an **I_SENDFD ioctl** over a stream pipe. *arg* is a pointer to a data buffer large enough to hold an **strrecvfd** data structure containing the following members:

```
int fd;
uid_t uid;
gid_t gid;
char fill[8];
```

fd is an integer file descriptor. **uid** and **gid** are the user ID and group ID, respectively, of the sending stream.

If **O_NDELAY** and **O_NONBLOCK** are clear [see **open**(2)], **I_RECVFD** will block until a message is present at the stream head. If **O_NDELAY** or **O_NONBLOCK** is set, **I_RECVFD** will fail with **errno** set to **EAGAIN** if no message is present at the stream head.

If the message at the stream head is a message sent by an **I_SENDFD**, a new user file descriptor is allocated for the file pointer contained in the message. The new file descriptor is placed in the **fd** field of the **strrecvfd** structure. The structure is copied into the user data buffer pointed to by *arg*.

On failure, **errno** is set to one of the following values:

EAGAIN	A message is not present at the stream head read queue, and the **O_NDELAY** or **O_NONBLOCK** flag is set.
EBADMSG	The message at the stream head read queue is not a message containing a passed file descriptor.
EFAULT	*arg* points outside the allocated address space.
EMFILE	**NOFILES** file descriptors are currently open.
ENXIO	Hangup received on *fildes*.
EOVERFLOW	*uid* or *gid* is too large to be stored in the structure pointed to by *arg*.

I_S_RECVFD Retrieves the file descriptor associated with the message sent by an **I_SENDFD ioctl** over a stream pipe. *arg* is a pointer to a data buffer large enough to hold an **s_strrecvfd** data structure containing the following members:

```
int fd;
uid_t uid;
gid_t gid;
struct sub_attr s_attrs;
```

fd is an integer file descriptor. **uid** and **gid** are the user ID and group ID, respectively, of the sending stream. **sub_attr** contains security relevant information. The **sub_attr** structure is used as an argument for the **secadvise**(2) system call, which provides advisory access information.

If **O_NDELAY** and **O_NONBLOCK** are clear [see **open**(2)], **I_RECVFD** will block until a message is present at the stream head. If **O_NDELAY** or **O_NONBLOCK** is set, **I_RECVFD** will fail with **errno** set to **EAGAIN** if no message is present at the stream head.

If the message at the stream head is a message sent by an **I_SENDFD**, a new user file descriptor is allocated for the file pointer contained in the message. The new file descriptor is placed in the **fd** field of the **s_strrecvfd** structure. The structure is copied into the user data buffer pointed to by *arg*.

On failure, **errno** is set to one of the following values:

ENOMEM	The system cannot allocate memory for the **s_strrecvfd** structure.
EAGAIN	A message is not present at the stream head read queue, and the **O_NDELAY** or **O_NONBLOCK** flag is set.
EBADMSG	The message at the stream head read queue is not a message containing a passed file descriptor.
EFAULT	*arg* points outside the allocated address space.
EMFILE	**NOFILES** file descriptors are currently open.
ENXIO	Hangup received on *fildes*.
EOVERFLOW	*uid* or *gid* is too large to be stored in the structure pointed to by *arg*.

I_LIST Allows the user to list all the module names on the stream, up to and including the topmost driver name. If *arg* is **NULL**, the return value is the number of modules, including the driver, that are on the stream pointed to by *fildes*. This allows the user to allocate enough space for the module names. If *arg* is non-**NULL**, it should point to an **str_list** structure that has the following members:

```
int sl_nmods;
struct str_mlist   *sl_modlist;
```

The **str_mlist** structure has the following member:

```
char l_name[FMNAMESZ+1];
```

sl_nmods indicates the number of entries the user has allocated in the array. On success, the return value is 0, **sl_modlist** contains the list of module names, and **sl_nmods** indicates the number of entries that have been filled in. On failure, **errno** may be set to one of the following values:

EINVAL	The **sl_nmods** member is less than 1.
EAGAIN	Unable to allocate buffers

I_ATMARK Allows the user to see if the current message on the stream head read queue is "marked" by some module downstream. *arg* determines how the checking is done when there may be multiple marked messages on the stream head read queue. It may take the following values:

ANYMARK	Check if the message is marked.
LASTMARK	Check if the message is the last one marked on the queue.

If both **ANYMARK** and **LASTMARK** are set, **ANYMARK** supersedes **LASTMARK**.

The return value is 1 if the mark condition is satisfied and 0 otherwise. On failure, **errno** may be set to the following value:

EINVAL	A value other than **(ANYMARK|LASTMARK)** is set in *arg*.

I_CKBAND Check if the message of a given priority band exists on the stream head read queue. This returns 1 if a message of a given priority exists, or –1 on error. *arg* should be an integer containing the value of the priority band in question. On failure, **errno** may be set to the following value:

EINVAL	Invalid *arg* value.

I_GETBAND Returns the priority band of the first message on the stream head read queue in the integer referenced by *arg*. On failure, **errno** may be set to the following value:

ENODATA	No message on the stream head read queue.

I_CANPUT Check if a certain band is writable. *arg* is set to the priority band in question. The return value is 0 if the priority band *arg* is flow controlled, 1 if the band is writable, or –1 on error. On failure, **errno** may be set to the following value:

EINVAL	Invalid *arg* value.

I_SETCLTIME

Allows the user to set the time the stream head will delay when a stream is closing and there is data on the write queues. Before closing each module and driver, the stream head will delay for the specified amount of time to allow the data to drain. If, after the delay, data is still present, data will be flushed. *arg* is a pointer to the number of milliseconds to delay, rounded up to the nearest valid value on the system. The default is fifteen seconds. On failure, **errno** may be set to the following value:

EINVAL	Invalid *arg* value.

I_GETCLTIME
Returns the close time delay in the long pointed by *arg*.

The following four commands are used for connecting and disconnecting multiplexed STREAMS configurations.

I_LINK Connects two streams, where *fildes* is the file descriptor of the stream connected to the multiplexing driver, and *arg* is the file descriptor of the stream connected to another driver. The stream designated by *arg* gets connected below the multiplexing driver. **I_LINK** requires the multiplexing driver to send an acknowledgement message to the stream head regarding the linking operation. This call returns a multiplexor ID number (an identifier used to disconnect the multiplexor, see **I_UNLINK**) on success, and a –1 on failure. On failure, **errno** is set to one of the following values:

ENXIO	Hangup received on *fildes*.
ETIME	Time out before acknowledgement message was received at stream head.
EAGAIN	Temporarily unable to allocate storage to perform the **I_LINK**.
ENOSR	Unable to allocate storage to perform the **I_LINK** due to insufficient STREAMS memory resources.
EBADF	*arg* is not a valid, open file descriptor.
EINVAL	*fildes* stream does not support multiplexing.
EINVAL	*arg* is not a stream, or is already linked under a multiplexor.
EINVAL	The specified link operation would cause a "cycle" in the resulting configuration; that is, if a given driver is linked into a multiplexing configuration in more than one place.
EINVAL	*fildes* is the file descriptor of a pipe or FIFO.

An **I_LINK** can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the stream head of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, **I_LINK** will fail with **errno** set to the value in the message.

I_UNLINK Disconnects the two streams specified by *fildes* and *arg*. *fildes* is the file descriptor of the stream connected to the multiplexing driver. *arg* is the multiplexor ID number that was returned by the **I_LINK**. If *arg* is –1, then all Streams which were linked to *fildes* are disconnected. As in **I_LINK**, this command requires the multiplexing driver to acknowledge the unlink. On failure, **errno** is set to one of the following values:

ENXIO	Hangup received on *fildes*.
ETIME	Time out before acknowledgement message was received at stream head.
ENOSR	Unable to allocate storage to perform the **I_UNLINK** due to insufficient STREAMS memory resources.
EINVAL	*arg* is an invalid multiplexor ID number or *fildes* is not the stream on which the **I_LINK** that returned *arg* was performed.
EINVAL	*fildes* is the file descriptor of a pipe or FIFO.

An **I_UNLINK** can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error or a hangup is received at the stream head of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, **I_UNLINK** will fail with **errno** set to the value in the message.

I_PLINK Connects two streams, where *fildes* is the file descriptor of the stream connected to the multiplexing driver, and *arg* is the file descriptor of the stream connected to another driver. The stream designated by *arg* gets connected via a persistent link below the multiplexing driver. **I_PLINK** requires the multiplexing driver to send an acknowledgement message to the stream head regarding the linking operation. This call creates a persistent link which can exist even if the file descriptor *fildes* associated with the upper stream to the multiplexing driver is closed. This call returns a multiplexor ID number (an identifier that may be used to disconnect the multiplexor, see **I_PUNLINK**) on success, and a –1 on failure. On failure, **errno** may be set to one of the following values:

ENXIO	Hangup received on *fildes*.
ETIME	Time out before acknowledgement message was received at the stream head.
EAGAIN	Unable to allocate STREAMS storage to perform the **I_PLINK**.
EBADF	*arg* is not a valid, open file descriptor.
EINVAL	*fildes* does not support multiplexing.
EINVAL	*arg* is not a stream or is already linked under a multiplexor.
EINVAL	The specified link operation would cause a "cycle" in the resulting configuration; that is, if a given stream head is linked into a multiplexing configuration in more than one place.

EINVAL *fildes* is the file descriptor of a pipe or FIFO.

An **I_PLINK** can also fail while waiting for the multiplexing driver to acknowledge the link request, if a message indicating an error on a hangup is received at the stream head of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, **I_PLINK** will fail with **errno** set to the value in the message.

I_PUNLINK Disconnects the two streams specified by *fildes* and *arg* that are connected with a persistent link. *fildes* is the file descriptor of the stream connected to the multiplexing driver. *arg* is the multiplexor ID number that was returned by **I_PLINK** when a stream was linked below the multiplexing driver. If *arg* is **MUXID_ALL** then all streams which are persistent links to *fildes* are disconnected. As in **I_PLINK**, this command requires the multiplexing driver to acknowledge the unlink. On failure, **errno** may be set to one of the following values:

ENXIO	Hangup received on *fildes*.
ETIME	Time out before acknowledgement message was received at the stream head.
EAGAIN	Unable to allocate buffers for the acknowledgement message.
EINVAL	Invalid multiplexor ID number.
EINVAL	*fildes* is the file descriptor of a pipe or FIFO.

An **I_PUNLINK** can also fail while waiting for the multiplexing driver to acknowledge the link request if a message indicating an error or a hangup is received at the stream head of *fildes*. In addition, an error code can be returned in the positive or negative acknowledgement message. For these cases, **I_PUNLINK** will fail with **errno** set to the value in the message.

SEE ALSO

close(2), **fcntl**(2), **getmsg**(2), **intro**(2), **ioctl**(2), **open**(2), **poll**(2), **putmsg**(2), **read**(2), **signal**(2), **signal**(5), **write**(2)

DIAGNOSTICS

Unless specified otherwise above, **ioctl** returns 0 on success and –1 on failure and sets **errno** as indicated.

NAME

`sw01` – Portable Device Interface (PDI) WORM Target Driver

DESCRIPTION

The **sw01** driver is a PDI-compliant WORM (Write Once Read Many) target driver that provides access to one or more WORM drives. Each drive must be attached to a SCSI Bus that is controlled by a PDI-compliant host adapter driver.

Access to the particular drive is accomplished through the **sw01** device nodes located in **/dev/[r]worm**.

Each device node identifies a particular drive based on the SCSI ID assigned to that drive. The binding between a device node and a WORM drive is as follows:

/dev/rworm/worm0	WORM drive with lowest SCSI ID
/dev/rworm/worm1	WORM drive with next to lowest SCSI ID

and so on.

A WORM drive uses removable media divided into consecutively numbered, fixed-size sectors that may be accessed in any order, similar to a hard disk. Most of the standard tools for reading and writing to and from a hard disk, such as such as **dd(1)** or **read(2)**, work with a WORM drive. However, keep in mind that for WORM drives, each sector can be written to only once. This characteristic causes problems if a WORM device is mounted [**mount**(1M)] without using the read-only flag, **-r**.

ioctl Calls

The **sw01** driver supports several **ioctl** functions [see **ioctl**(2) in the *Programmer's Reference Manual*], which are accessed through the **ioctl** system call. Many of the supported **ioctl** calls provide a convenient method for sending one of the preselected SCSI commands directly to the drive. SCSI commands not explicitly supported by **sw01** can be sent to the drive using the pass-through facility provided by the SDI host adapter driver.

The following **ioctl** calls are defined and required by the SDI interface.

B_GETTYPE	Returns the type of peripheral bus (for example, **scsi**) used and the name of the driver (for example, **sw01**) for this specific device
B_GETDEV	Returns the major and minor number of the pass-through device for the WORM drive
SDI_RESERVE	Sends a SCSI **Reserve** command to the drive
SDI_RELEASE	Sends a SCSI **Release** command to the drive

The following **ioctl** calls send the appropriate Group-0 SCSI command to the device. These commands are defined by the SCSI bus specification and should also be described in the SCSI interface section of the reference manual supplied with your WORM drive.

W_TESTUNIT	Sends a Test Unit Ready command to the device

W_REZERO	Sends a Rezero Device command to the device
W_SEEK	Sends a Seek command to the device
W_INQUIR	Sends an Inquiry command to the device, and returns the resulting data back to the calling process
W_STARTUNIT	Sends a Start Unit command to the device
W_STOPUNIT	Sends a Stop Unit command to the device
W_PREVMV	Sends a Prevent Media Removal command to the device
W_ALLOMV	Sends an Allow Media Removal command to the device

The following **ioctl** calls send the appropriate Group-1 SCSI command to the device. The Group-1 SCSI commands are defined by the SCSI bus specification and should be described in the SCSI interface section of the reference manual supplied with your WORM drive.

W_READCAPA	Sends a Read Capacity command to the device, and returns the data sent by the drive
W_VERIFY	Sends a Verify command to the device

The following **ioctl** calls send the appropriate Group-6 SCSI command to the drive. Group-6 SCSI commands are vendor specific and should be described in the SCSI interface section of the reference manual supplied with your drive. Since the format of these SCSI commands is vendor specific, these **ioctl** calls are supported only by products compatible with the Toshiba D070 drive.

W_STNCHECK	Sends a Stand-By Check command to the device
W_LOADCART	Sends a Load Cartridge command to the device
W_UNLOADCA	Sends an Unload Cartridge command to the device
W_READCB	Sends a Read Control Block command to the device

The following **ioctl** calls send the appropriate Group-7 SCSI command to the drive. Group-7 SCSI commands are vendor specific and should be described in the SCSI interface section of the reference manual supplied with your drive. Since the format of these SCSI commands is vendor specific, these **ioctl** calls are supported only by products compatible with the Toshiba D070 drive.

W_CHECK	Sends a Check command to the device
W_CCHECK	Sends a Contrary Check command to the device

The following **ioctl** calls are also supported the **sw01** driver.

B_GET_SUBDEVS	Returns the number of sub-devices supported by this driver
W_ERRMSGON	Enables the **sw01** related system error messages
W_ERRMSGOFF	Disables the **sw01** related system error messages

Files

/usr/include/sys/sw01.h
/etc/conf/pack.d/sw01/space.c
/dev/[r]worm/*

```
/usr/include/sys/scsi.h
/usr/include/sys/sdi.h
/usr/include/sys/sdi_edt.h
```

REFERENCES

adsc(7), dpt(7), ioctl(2), mcis(7), mount(1M), sc01(7), sd01(7), st01(7), wd7000(7)

NAME

sxt – pseudo-device driver

DESCRIPTION

The special file **/dev/sxt** is a pseudo-device driver that interposes a discipline between the standard **tty** line disciplines and a real device driver. The standard disciplines manipulate virtual tty structures (channels) declared by the **/dev/sxt** driver. **/dev/sxt** acts as a discipline manipulating a real tty structure declared by a real device driver. The **/dev/sxt** driver is currently only used by the **shl**(1) command.

Virtual ttys are named by inodes in the subdirectory **/dev/sxt** and are allocated in groups of up to eight. To allocate a group, a program should exclusively open a file with a name of the form **/dev/sxt/??0** (channel 0) and then execute a **SXTIOCLINK** **ioctl** call to initiate the multiplexing.

Only one channel, the controlling channel, can receive input from the keyboard at a time; others attempting to read will be blocked.

There are two groups of **ioctl**(2) commands supported by **sxt**. The first group contains the standard **ioctl** commands described in **termio**(7), with the addition of the following:

TIOCEXCL	Set exclusive use mode: no further opens are permitted until the file has been closed.
TIOCNXCL	Reset exclusive use mode: further opens are once again permitted.

The second group are commands to **sxt** itself. Some of these may only be executed on channel 0.

SXTIOCLINK Allocate a channel group and multiplex the virtual ttys onto the real tty. The argument is the number of channels to allocate. This command may only be executed on channel 0. Possible errors include:

EINVAL	The argument is out of range.
ENOTTY	The command was not issued from a real tty.
ENXIO	**linesw** is not configured with **sxt**.
EBUSY	An **SXTIOCLINK** command has already been issued for this real **tty**.
ENOMEM	There is no system memory available for allocating the virtual tty structures.
EBADF	Channel 0 was not opened before this call.

SXTIOCSWTCH Set the controlling channel. Possible errors include:

EINVAL	An invalid channel number was given.
EPERM	The command was not executed from channel 0.

`SXTIOCWF`	Cause a channel to wait until it is the controlling channel. This command will return the error, **`EINVAL`**, if an invalid channel number is given.
`SXTIOCUBLK`	Turn off the **`loblk`** control flag in the virtual tty of the indicated channel. The error **`EINVAL`** will be returned if an invalid number or channel 0 is given.
`SXTIOCSTAT`	Get the status (blocked on input or output) of each channel and store in the **`sxtblock`** structure referenced by the argument. The error **`EFAULT`** will be returned if the structure cannot be written.
`SXTIOCTRACE`	Enable tracing. Tracing information is written to the console. This command has no effect if tracing is not configured.
`SXTIOCNOTRACE`	Disable tracing. This command has no effect if tracing is not configured.

FILES

`/dev/sxt/??[0-7]`	Virtual tty devices

SEE ALSO

`ioctl`(2), **`open`**(2), **`shl`**(1), **`stty`**(1) **`termio`**(7)

NAME

TCP – Internet Transmission Control Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_STREAM, 0);

t = t_open("/dev/tcp", O_RDWR);
```

DESCRIPTION

TCP is the virtual circuit protocol of the Internet protocol family. It provides reliable, flow-controlled, in order, two-way transmission of data. It is a byte-stream protocol layered above the Internet Protocol (IP), the Internet protocol family's internetwork datagram delivery protocol.

Programs can access TCP using the socket interface as a **SOCK_STREAM** socket type, or using the Transport Level Interface (TLI) where it supports the connection-oriented (**T_COTS_ORD**) service type.

TCP uses IP's host-level addressing and adds its own per-host collection of port addresses. The endpoints of a TCP connection are identified by the combination of an IP address and a TCP port number. Although other protocols, such as the User Datagram Protocol (UDP), may use the same host and port address format, the port space of these protocols is distinct. See **inet**(7) for details on the common aspects of addressing in the Internet protocol family.

Sockets utilizing TCP are either active or passive. Active sockets initiate connections to passive sockets. Both types of sockets must have their local IP address and TCP port number bound with the **bind**(3N) system call after the socket is created. By default, TCP sockets are active. A passive socket is created by calling the **listen**(3N) system call after binding the socket with **bind()**. This establishes a queuing parameter for the passive socket. After this, connections to the passive socket can be received with the **accept**(3N) system call. Active sockets use the **connect**(3N) call after binding to initiate connections.

By using the special value **INADDR_ANY**, the local IP address can be left unspecified in the **bind()** call by either active or passive TCP sockets. This feature is usually used if the local address is either unknown or irrelevant. If left unspecified, the local IP address will be bound at connection time to the address of the network interface used to service the connection.

Once a connection has been established, data can be exchanged using the **read**(2) and **write**(2) system calls.

TCP supports one socket option, **TCP_NODELAY**, which is set with **setsockopt()** and tested with **getsockopt**(3N). Under most circumstances, TCP sends data when it is presented. When outstanding data has not yet been acknowledged, it gathers small amounts of output to be sent in a single packet once an acknowledgement is received. For a small number of clients, such as window systems that send a stream of mouse events which receive no replies, this packetization may cause significant delays. Therefore, TCP provides a boolean option, **TCP_NODELAY** (defined in **/usr/include/netinet/tcp.h**), to defeat this algorithm. The option level for the **setsockopt()** call is the protocol number for TCP, available from **getprotobyname()** [see **getprotoent**(3N)].

Options at the IP level may be used with TCP; See **ip**(7).

TCP provides an urgent data mechanism, which may be invoked using the out-of-band provisions of **send**(3N). The caller may mark one byte as urgent with the **MSG_OOB** flag to **send**(3N). This sets an urgent pointer pointing to this byte in the TCP stream. The receiver on the other side of the stream is notified of the urgent data by a **SIGURG** signal. The **SIOCATMARK ioctl()** request returns a value indicating whether the stream is at the urgent mark. Because the system never returns data across the urgent mark in a single **read**(2) call, it is possible to advance to the urgent data in a simple loop which reads data, testing the socket with the **SIOCATMARK ioctl()** request, until it reaches the mark.

Incoming connection requests that include an IP source route option are noted, and the reverse source route is used in responding.

A checksum over all data helps TCP implement reliability. Using a window-based flow control mechanism that makes use of positive acknowledgements, sequence numbers, and a retransmission strategy, TCP can usually recover when datagrams are damaged, delayed, duplicated or delivered out of order by the underlying communication medium.

If the local TCP receives no acknowledgements from its peer for a period of time, as would be the case if the remote machine crashed, the connection is closed and an error is returned to the user. If the remote machine reboots or otherwise loses state information about a TCP connection, the connection is aborted and an error is returned to the user.

SEE ALSO

accept(3N), **bind**(3N), **connect**(3N), **getprotoent**(3N), **getsockopt**(3N), **inet**(7), **ip**(7), **listen**(3N), **read**(2), **send**(3N), **write**(2)

Postel, Jon, *Transmission Control Protocol - DARPA Internet Program Protocol Specification*, RFC 793, Network Information Center, SRI International, Menlo Park, Calif., September 1981

DIAGNOSTICS

A socket operation may fail if:

EISCONN	A **connect()** operation was attempted on a socket on which a **connect()** operation had already been performed.
ETIMEDOUT	A connection was dropped due to excessive retransmissions.
ECONNRESET	The remote peer forced the connection to be closed (usually because the remote machine has lost state information about the connection due to a crash).
ECONNREFUSED	The remote peer actively refused connection establishment (usually because no process is listening to the port).
EADDRINUSE	A **bind()** operation was attempted on a socket with a network address/port pair that has already been bound to another socket.

EADDRNOTAVAIL	A **bind()** operation was attempted on a socket with a network address for which no network interface exists.
EACCES	A **bind()** operation was attempted with a reserved port number and the effective user ID of the process was not the privileged user.
ENOBUFS	The system ran out of memory for internal data structures.

NAME

`termio` – general terminal interface

SYNOPSIS

```
#include <termio.h>

ioctl(int fildes, int request, struct termio *arg);
ioctl(int fildes, int request, int arg);

#include <termios.h>

ioctl(int fildes, int request, struct termios *arg);
```

DESCRIPTION

System V supports a general interface for asynchronous communications ports that is hardware-independent. The user interface to this functionality is via function calls (the preferred interface) described in **termios**(2) or **ioctl** commands described in this section. This section also discusses the common features of the terminal subsystem which are relevant with both user interfaces.

When a terminal file is opened, it normally causes the process to wait until a connection is established. In practice, users' programs seldom open terminal files; they are opened by the system and become a user's standard input, output, and error files. The very first terminal file opened by the session leader, which is not already associated with a session, becomes the controlling terminal for that session. The controlling terminal plays a special role in handling quit and interrupt signals, as discussed below. The controlling terminal is inherited by a child process during a **fork**(2). A process can break this association by changing its session using **setsid**(2).

A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters may be typed at any time, even while output is occurring, and are only lost when the character input buffers of the system become completely full, which is rare (for example, if the number of characters in the line discipline buffer exceeds **{MAX_CANON}** and **IMAXBEL** [see below] is not set), or when the user has accumulated {**MAX_INPUT**} number of input characters that have not yet been read by some program. When the input limit is reached, all the characters saved in the buffer up to that point are thrown away without notice.

Session Management (Job Control)

A control terminal will distinguish one of the process groups in the session associated with it to be the foreground process group. All other process groups in the session are designated as background process groups. This foreground process group plays a special role in handling signal-generating input characters, as discussed below. By default, when a controlling terminal is allocated, the controlling process's process group is assigned as foreground process group.

Background process groups in the controlling process's session are subject to a job control line discipline when they attempt to access their controlling terminal. Process groups can be sent signals that will cause them to stop, unless they have made other arrangements. An exception is made for members of orphaned process groups. These are process groups which do not have a member with a parent in another process group that is in the same session and therefore shares the same controlling terminal. When a member's orphaned process group attempts to access its controlling terminal, errors will be returned. since there is no process to continue it if it should stop.

If a member of a background process group attempts to read its controlling terminal, its process group will be sent a **SIGTTIN** signal, which will normally cause the members of that process group to stop. If, however, the process is ignoring or holding **SIGTTIN**, or is a member of an orphaned process group, the read will fail with **errno** set to **EIO**, and no signal will be sent.

If a member of a background process group attempts to write its controlling terminal and the **TOSTOP** bit is set in the **c_lflag** field, its process group will be sent a **SIGTTOU** signal, which will normally cause the members of that process group to stop. If, however, the process is ignoring or holding **SIGTTOU**, the write will succeed. If the process is not ignoring or holding **SIGTTOU** and is a member of an orphaned process group, the write will fail with **errno** set to **EIO**, and no signal will be sent.

If **TOSTOP** is set and a member of a background process group attempts to **ioctl** its controlling terminal, and that **ioctl** will modify terminal parameters (for example, **TCSETA**, **TCSETAW**, **TCSETAF**, or **TIOCSPGRP**), its process group will be sent a **SIGTTOU** signal, which will normally cause the members of that process group to stop. If, however, the process is ignoring or holding **SIGTTOU**, the ioctl will succeed. If the process is not ignoring or holding **SIGTTOU** and is a member of an orphaned process group, the write will fail with **errno** set to **EIO**, and no signal will be sent.

Canonical Mode Input Processing

Normally, terminal input is processed in units of lines. A line is delimited by a newline (ASCII LF) character, an end-of-file (ASCII EOT) character, or an end-of-line character. This means that a program attempting to read will be suspended until an entire line has been typed. Also, no matter how many characters are requested in the read call, at most one line will be returned. It is not necessary, however, to read a whole line at once; any number of characters may be requested in a read, even one, without losing information.

During input, erase and kill processing is normally done. The ERASE character (by default, the character #) erases the last character typed. The WERASE character (the character control-W) erases the last "word" typed in the current input line (but not any preceding spaces or tabs). A "word" is defined as a sequence of non-blank characters, with tabs counted as blanks. Neither ERASE nor WERASE will erase beyond the beginning of the line. The KILL character (by default, the character @) kills (deletes) the entire input line, and optionally outputs a newline character. All these characters operate on a key stroke basis, independent of any backspacing or tabbing that may have been done. The REPRINT character (the character control-R) prints a newline followed by all characters that have not been read. Reprinting also occurs automatically if characters that would normally be erased from the screen are fouled by program output. The characters are reprinted as if they were being echoed; consequencely, if **ECHO** is not set, they are not printed.

The ERASE and KILL characters may be entered literally by preceding them with the escape character (\ ). In this case, the escape character is not read. The erase and kill characters may be changed.

Non-canonical Mode Input Processing

In non-canonical mode input processing, input characters are not assembled into lines, and erase and kill processing does not occur. The **MIN** and **TIME** values are used to determine how to process the characters received.

MIN represents the minimum number of characters that should be received when the read is satisfied (that is, when the characters are returned to the user). **TIME** is a timer of 0.10-second granularity that is used to timeout bursty and short-term data transmissions. The values for **MIN** and **TIME** should be set by the programmer in the **termios** or **termio** structure. The four possible values for **MIN** and **TIME** and their interactions are described below.

Case A: **MIN** > 0, **TIME** > 0

In this case, **TIME** serves as an intercharacter timer and is activated after the first character is received. Since it is an intercharacter timer, it is reset after a character is received. The interaction between **MIN** and **TIME** is as follows: as soon as one character is received, the intercharacter timer is started. If **MIN** characters are received before the intercharacter timer expires (note that the timer is reset upon receipt of each character), the read is satisfied. If the timer expires before **MIN** characters are received, the characters received to that point are returned to the user. Note that if **TIME** expires, at least one character will be returned because the timer would not have been enabled unless a character was received. In this case (**MIN** > 0, **TIME** > 0), the read sleeps until the **MIN** and **TIME** mechanisms are activated by the receipt of the first character. If the number of characters read is less than the number of characters available, the timer is not reactivated and the subsequent read is satisfied immediately.

Case B: **MIN** > 0, **TIME** = 0

In this case, since the value of **TIME** is zero, the timer plays no role and only **MIN** is significant. A pending read is not satisfied until **MIN** characters are received (the pending read sleeps until **MIN** characters are received). A program that uses this case to read record based terminal I/O may block indefinitely in the read operation.

Case C: **MIN** = 0, **TIME** > 0

In this case, since **MIN** = 0, **TIME** no longer represents an intercharacter timer: it now serves as a read timer that is activated as soon as a **read** is done. A read is satisfied as soon as a single character is received or the read timer expires. Note that, in this case, if the timer expires, no character is returned. If the timer does not expire, the only way the read can be satisfied is if a character is received. In this case, the read will not block indefinitely waiting for a character; if no character is received within **TIME***.10 seconds after the read is initiated, the read returns with zero characters.

Case D: **MIN** = 0, **TIME** = 0

In this case, return is immediate. The minimum of either the number of characters requested or the number of characters currently available is returned without waiting for more characters to be input.

Comparison of the Different Cases of MIN, TIME Interaction

Some points to note about **MIN** and **TIME**:

1. In the following explanations, note that the interactions of **MIN** and **TIME** are not symmetric. For example, when **MIN** > 0 and **TIME** = 0, **TIME** has no effect. However, in the opposite case, where **MIN** = 0 and **TIME** > 0, both **MIN** and **TIME** play a role in that **MIN** is satisfied with the receipt of a single character.
2. Also note that in case A (**MIN** > 0, **TIME** > 0), **TIME** represents an intercharacter timer, whereas in case C (**TIME** = 0, **TIME** > 0), **TIME** represents a read timer.

These two points highlight the dual purpose of the **MIN/TIME** feature. Cases A and B, where **MIN** > 0, exist to handle burst mode activity (for example, file transfer programs), where a program would like to process at least **MIN** characters at a time. In case A, the intercharacter timer is activated by a user as a safety measure; in case B, the timer is turned off.

Cases C and D exist to handle single character, timed transfers. These cases are readily adaptable to screen-based applications that need to know if a character is present in the input queue before refreshing the screen. In case C, the read is timed, whereas in case D, it is not.

Another important note is that **MIN** is always just a minimum. It does not denote a record length. For example, if a program does a read of 20 bytes, **MIN** is 10, and 25 characters are present, then 20 characters will be returned to the user.

Writing Characters

When one or more characters are written, they are transmitted to the terminal as soon as previously written characters have finished typing. Input characters are echoed as they are typed if echoing has been enabled. If a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds some limit. When the queue is drained down to some threshold, the program is resumed.

Special Characters

Certain characters have special functions on input. These functions and their default character values are summarized as follows:

INTR (Rubout or ASCII DEL) generates a **SIGINT** signal. **SIGINT** is sent to all frequent processes associated with the controlling terminal. Normally, each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed upon location. [See **signal**(5)].

QUIT (CTRL-| or ASCII FS) generates a **SIGQUIT** signal. Its treatment is identical to the interrupt signal except that, unless a receiving process has made other arrangements, it will not only be terminated but a core image file (called **core**) will be created in the current working directory.

ERASE (#) erases the preceding character. It does not erase beyond the start of a line, as delimited by a NL, EOF, EOL, or EOL2 character.

WERASE (CTRL-W or ASCII ETX) erases the preceding "word". It does not erase beyond the start of a line, as delimited by a NL, EOF, EOL, or EOL2 character.

KILL	(@) deletes the entire line, as delimited by a NL, EOF, EOL, or EOL2 character.
REPRINT	(CTRL-R or ASCII DC2) reprints all characters, preceded by a newline, that have not been read.
EOF	(CTRL-D or ASCII EOT) may be used to generate an end-of-file from a terminal. When received, all the characters waiting to be read are immediately passed to the program, without waiting for a newline, and the EOF is discarded. Thus, if no characters are waiting (that is, the EOF occurred at the beginning of a line) zero characters are passed back, which is the standard end-of-file indication. The EOF character is not echoed unless it is escaped or ECHOCTL is set. Because EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.
NL	(ASCII LF) is the normal line delimiter. It cannot be changed or escaped.
EOL	(ASCII NULL) is an additional line delimiter, like NL. It is not normally used.
EOL2	is another additional line delimiter.
SWTCH	(CTRL-Z or ASCII EM) is used only when **shl** layers is invoked.
SUSP	(CTRL-Z or ASCII SUB) generates a **SIGTSTP** signal. **SIGTSTP** stops all processes in the foreground process group for that terminal.
DSUSP	(CTRL-Y or ASCII EM) It generates a **SIGTSTP** signal as SUSP does, but the signal is sent when a process in the foreground process group attempts to read the DSUSP character, rather than when it is typed.
STOP	(CTRL-S or ASCII DC3) can be used to suspend output temporarily. It is useful with CRT terminals to prevent output from disappearing before it can be read. While output is suspended, STOP characters are ignored and not read.
START	(CTRL-Q or ASCII DC1) is used to resume output. Output has been suspended by a STOP character. While output is not suspended, START characters are ignored and not read.
DISCARD	(CTRL-O or ASCII SI) causes subsequent output to be discarded. Output is discarded until another DISCARD character is typed, more input arrives, or the condition is cleared by a program.
LNEXT	(CTRL-V or ASCII SYN) causes the special meaning of the next character to be ignored. This works for all the special characters mentioned above. It allows characters to be input that would otherwise be interpreted by the system (for example, KILL, QUIT).

The character values for INTR, QUIT, ERASE, WERASE, KILL, REPRINT, EOF, EOL, EOL2, SWTCH, SUSP, DSUSP, STOP, START, DISCARD, and LNEXT may be changed to suit individual tastes. If the value of a special control character is _POSIX_VDISABLE (0), the function of that special control character is disabled. The ERASE, KILL, and EOF characters may be escaped by a preceding \ character, in which case no special function is done. Any of the special characters may be preceded by the LNEXT character, in which case no special function is done.

Modem Disconnect

When a modem disconnect is detected, a **SIGHUP** signal is sent to the terminal's controlling process. Unless other arrangements have been made, these signals cause the process to terminate. If **SIGHUP** is ignored or caught, any subsequent read returns with an end-of-file indication until the terminal is closed.

Processes in background process groups that attempt to access the controlling terminal after modem disconnect while the terminal is still allocated to the session will receive appropriate **SIGTTOU** and **SIGTTIN** signals. Unless other arrangements have been made, this signal causes the processes to stop.

The controlling terminal will remain in this state until it is reinitialized with a successful open by the controlling process, or deallocated by the controlling process.

Terminal Parameters

The parameters that control the behavior of devices and modules providing the **termios** interface are specified by the **termios** structure defined by **termios.h**. Several **ioctl**(2) system calls that fetch or change these parameters use this structure that contains the following members:

```
tcflag_t   c_iflag;           /* input modes */
tcflag_t   c_oflag;           /* output modes */
tcflag_t   c_cflag;           /* control modes */
tcflag_t   c_lflag;           /* local modes */
cc_t       c_cc[NCCS];        /* control chars */
```

The special control characters are defined by the array **c_cc**. The symbolic name **NCCS** is the size of the control-character array and is also defined by **termios.h**. The relative positions, subscript names, and typical default values for each function are as follows:

0	**VINTR**	DEL
1	**VQUIT**	FS
2	**VERASE**	#
3	**VKILL**	@
4	**VEOF**	EOT
5	**VEOL**	NUL
6	**VEOL2**	NUL
7	**VSWTCH**	NUL
8	**VSTRT**	DC1
9	**VSTOP**	DC3
10	**VSUSP**	SUB
11	**VDSUSP**	EM
12	**VREPRINT**	DC2
13	**VDISCRD**	SI
14	**VWERASE**	ETB
15	**VLNEXT**	SYN
16–19	reserved	

For the non-canonical mode the positions of **VEOF** and **VEOL** are shared by **VMIN** and **VTIME**:

4	**VMIN**	used to set the value of MIN
5	**VTIME**	used to set the value of TIME

Input Modes

The `c_iflag` field describes the basic terminal input control:

IGNBRK	Ignore break condition.
BRKINT	Signal interrupt on break.
IGNPAR	Ignore characters with parity errors.
PARMRK	Mark parity errors.
INPCK	Enable input parity check.
ISTRIP	Strip character.
INLCR	Map NL to CR on input.
IGNCR	Ignore CR.
ICRNL	Map CR to NL on input.
IUCLC	Map upper-case to lower-case on input.
IXON	Enable start/stop output control.
IXANY	Enable any character to restart output.
IXOFF	Enable start/stop input control.
IMAXBEL	Echo BEL on input line too long.

If **IGNBRK** is set, a break condition (a character framing error with data all zeros) detected on input is ignored, that is, not put on the input queue and therefore not read by any process. If **IGNBRK** is not set and **BRKINT** is set, the break condition shall flush the input and output queues and if the terminal is the controlling terminal of a foreground process group, the break condition generates a single **SIGINT** signal to that foreground process group. If neither **IGNBRK** nor **BRKINT** is set, a break condition is read as a single ASCII NULL character ('\0'), or if **PARMRK** is set, as '\377', '\0', '\0'.

If **IGNPAR** is set, a byte with framing or parity errors (other than break) is ignored.

If **PARMRK** is set, and **IGNPAR** is not set, a byte with a framing or parity error (other than break) is given to the application as the three-character sequence: '\377', '\0', X, where X is the data of the byte received in error. To avoid ambiguity in this case, if **ISTRIP** is not set, a valid character of '\377' is given to the application as '\377', '\377'. If neither **IGNPAR** nor **PARMRK** is set, a framing or parity error (other than break) is given to the application as a single ASCII NULL character ('\0').

If **INPCK** is set, input parity checking is enabled. If **INPCK** is not set, input parity checking is disabled. This allows output parity generation without input parity errors. Note that whether input parity checking is enabled or disabled is independent of whether parity detection is enabled or disabled. If parity detection is enabled but input parity checking is disabled, the hardware to which the terminal is connected will recognize the parity bit, but the terminal special file will not check whether this is set correctly or not.

If **ISTRIP** is set, valid input characters are first stripped to seven bits, otherwise all eight bits are processed.

If **INLCR** is set, a received NL character is translated into a CR character. If **IGNCR** is set, a received CR character is ignored (not read). Otherwise, if **ICRNL** is set, a received CR character is translated into a NL character.

If **IUCLC** is set, a received upper case, alphabetic character is translated into the corresponding lower case character.

If **IXON** is set, start/stop output control is enabled. A received STOP character suspends output and a received START character restarts output. The STOP and START characters will not be read, but will merely perform flow control functions. If **IXANY** is set, any input character restarts output that has been suspended.

If **IXOFF** is set, the system transmits a STOP character when the input queue is nearly full, and a START character when enough input has been read so that the input queue is nearly empty again.

If **IMAXBEL** is set, the ASCII BEL character is echoed if the input stream overflows. Further input is not stored, but any input already present in the input stream is not disturbed. If **IMAXBEL** is not set, no BEL character is echoed, and all input present in the input queue is discarded if the input stream overflows.

The initial input control value is **BRKINT, ICRNL, IXON, ISTRIP**.

Output Modes

The **c_oflag** field specifies the system treatment of output:

OPOST	Post-process output.
OLCUC	Map lower case to upper on output.
ONLCR	Map NL to CR-NL on output.
OCRNL	Map CR to NL on output.
ONOCR	No CR output at column 0.
ONLRET	NL performs CR function.
OFILL	Use fill characters for delay.
OFDEL	Fill is DEL, else **NULL**.
NLDLY	Select newline delays:
NL0	
NL1	
CRDLY	Select carriage-return delays:
CR0	
CR1	
CR2	
CR3	
TABDLY	Select horizontal tab delays:
TAB0	or tab expansion:
TAB1	
TAB2	
TAB3	Expand tabs to spaces.
XTABS	Expand tabs to spaces.
BSDLY	Select backspace delays:
BS0	
BS1	
VTDLY	Select vertical tab delays:
VT0	
VT1	
FFDLY	Select form feed delays:
FF0	
FF1	

If **OPOST** is set, output characters are post-processed as indicated by the remaining flags; otherwise, characters are transmitted without change.

If **OLCUC** is set, a lower case alphabetic character is transmitted as the corresponding upper case character. This function is often used in conjunction with **IUCLC**.

If **ONLCR** is set, the NL character is transmitted as the CR-NL character pair. If **OCRNL** is set, the CR character is transmitted as the NL character. If **ONOCR** is set, no CR character is transmitted when at column 0 (first position). If **ONRET** is set, the NL character is assumed to do the carriage-return function; the column pointer is set to 0 and the delays specified for CR are used. Otherwise, the NL character is assumed to do just the line-feed function; the column pointer remains unchanged. The column pointer is also set to 0 if the CR character is actually transmitted.

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases, a value of 0 indicates no delay. If **OFILL** is set, fill characters are transmitted for delay instead of a timed delay. This is useful for high baud rate terminals that need only a minimal delay. If **OFDEL** is set, the fill character is DEL; otherwise it is **NULL**.

If a form-feed or vertical-tab delay is specified, it lasts for about 2 seconds.

Newline delay lasts about 0.10 seconds. If **ONLRET** is set, the carriage-return delays are used instead of the newline delays. If **OFILL** is set, two fill characters are transmitted.

Carriage-return delay type 1 is dependent on the current column position, type 2 is about 0.10 seconds, and type 3 is about 0.15 seconds. If **OFILL** is set, delay type 1 transmits two fill characters, and type 2 transmits four fill characters.

Horizontal-tab delay type 1 is dependent on the current column position. Type 2 is about 0.10 seconds. Type 3 specifies that tabs are to be expanded into spaces. If **OFILL** is set, two fill characters are transmitted for any delay.

Backspace delay lasts about 0.05 seconds. If **OFILL** is set, one fill character is transmitted.

The actual delays depend on line speed and system load.

The initial output control value is **OPOST, ONLCR, TAB3**.

Control Modes

The **c_cflag** field describes the hardware control of the terminal:

CBAUD	Baud rate:
B0	Hang up
B50	50 baud
B75	75 baud
B110	110 baud
B134	134 baud
B150	150 baud
B200	200 baud
B300	300 baud
B600	600 baud
B1200	1200 baud
B1800	1800 baud

B2400	2400 baud
B4800	4800 baud
B9600	9600 baud
B19200	19200 baud
EXTA	External A
B38400	38400 baud
EXTB	External B

CSIZE	Character size:
CS5	5 bits
CS6	6 bits
CS7	7 bits
CS8	8 bits

CSTOPB	Send two stop bits, else one
CREAD	Enable receiver
PARENB	Parity enable
PARODD	Odd parity, else even
HUPCL	Hang up on last close
CLOCAL	Local line, else dial-up
CIBAUD	Input baud rate, if different from output rate
PAREXT	Extended parity for mark and space parity

The **CBAUD** bits specify the baud rate. The zero baud rate, B0, is used to hang up the connection. If B0 is specified, the data-terminal-ready signal is not asserted. Normally, this disconnects the line. If the **CIBAUD** bits are not zero, they specify the input baud rate, with the **CBAUD** bits specifying the output baud rate; otherwise, the output and input baud rates are both specified by the **CBAUD** bits. The values for the **CIBAUD** bits are the same as the values for the **CBAUD** bits, shifted left **IBSHIFT** bits. For any particular hardware, impossible speed changes are ignored.

The **CSIZE** bits specify the character size in bits for both transmission and reception. This size does not include the parity bit, if any. If **CSTOPB** is set, two stop bits are used; otherwise, one stop bit is used. For example, at 110 baud, two stops bits are required.

If **PARENB** is set, parity generation and detection is enabled, and a parity bit is added to each character. If parity is enabled, the **PARODD** flag specifies odd parity if set; otherwise, even parity is used.

If **CREAD** is set, the receiver is enabled. Otherwise, no characters are received.

If **HUPCL** is set, the line is disconnected when the last process with the line open closes it or terminates. That is, the data-terminal-ready signal is not asserted.

If **CLOCAL** is set, the line is assumed to be a local, direct connection with no modem control; otherwise, modem control is assumed.

The initial hardware control value after open is **B300, CS8, CREAD, HUPCL**.

Local Modes

The **c_lflag** field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline provides the following:

ISIG	Enable signals.
ICANON	Canonical input (erase and kill processing).
XCASE	Canonical upper/lower presentation.
ECHO	Enable echo.
ECHOE	Echo erase character as BS-SP-BS.
ECHOK	Echo NL after kill character.
ECHONL	Echo NL.
NOFLSH	Disable flush after interrupt or quit.
TOSTOP	Send **SIGTTOU** for background output.
ECHOCTL	Echo control characters as ^*char*, delete as ^?.
ECHOPRT	Echo erase character as character erased.
ECHOKE	BS-SP-BS erase entire line on line kill.
FLUSHO	Output is being flushed.
PENDIN	Retype pending input at next read or input character.
IEXTEN	Enable extended (implementation-defined) functions.

If **ISIG** is set, each input character is checked against the special control characters INTR, QUIT, SWTCH, SUSP, STATUS, and DSUSP. If an input character matches one of these control characters, the function associated with that character is performed. If **ISIG** is not set, no checking is done. Thus, these special input functions are possible only if **ISIG** is set.

If **ICANON** is set, canonical processing is enabled. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, EOL, and EOL2. If **ICANON** is not set, read requests are satisfied directly from the input queue. A read is not satisfied until at least **MIN** characters have been received or the timeout value **TIME** has expired between characters. This allows fast bursts of input to be read efficiently while still allowing single character input. The time value represents tenths of seconds.

If **XCASE** is set, and if **ICANON** is set, an upper case letter is accepted on input by preceding it with a \ character, and is output preceded by a \ character. In this mode, the following escape sequences are generated on output and accepted on input:

for:	use:
`	\\'
|	\\!
~	\\^
{	\\(
}	\\)
\\	\\\\

For example, **A** is input as **\a**, **\n** as **\\n**, and **\N** as **\\\n**.

If **ECHO** is set, characters are echoed as received.

When **ICANON** is set, the following echo functions are possible.

1. If **ECHO** and **ECHOE** are set, and **ECHOPRT** is not set, the ERASE and WERASE characters are echoed as one or more ASCII BS SP BS, which clears the last character(s) from a CRT screen.

2. If **ECHO** and **ECHOPRT** are set, the first ERASE and WERASE character in a sequence echoes as a backslash (\), followed by the characters being erased. Subsequent ERASE and WERASE characters echo the characters being erased, in reverse order. The next non-erase character causes a slash (/) to be typed before it is echoed. **ECHOPRT** should be used for hard copy terminals.
3. If **ECHOKE** is set, the kill character is echoed by erasing each character on the line from the screen (using the mechanism selected by **ECHOE** and **ECHOPRT**).
4. If **ECHOK** is set, and **ECHOKE** is not set, the NL character is echoed after the kill character to emphasize that the line is deleted. Note that an escape character (\) or an LNEXT character preceding the erase or kill character removes any special function.
5. If **ECHONL** is set, the NL character is echoed even if **ECHO** is not set. This is useful for terminals set to local echo (so called half-duplex).

If **ECHOCTL** is set, all control characters (characters with codes between 0 and 37 octal) other than ASCII TAB, ASCII NL, the START character, and the STOP character, ASCII CR, and ASCII BS are echoed as ^**X**, where **X** is the character given by adding 100 octal to the code of the control character (so that the character with octal code 1 is echoed as **^A**), and the ASCII DEL character, with code 177 octal, is echoed as ^?.

If **NOFLSH** is set, the normal flush of the input and output queues associated with the INTR, QUIT, and SUSP characters is not done. This bit should be set when restarting system calls that read from or write to a terminal [see **sigaction**(2)].

If **TOSTOP** is set, the signal **SIGTTOU** is sent to a process that tries to write to its controlling terminal if it is not in the foreground process group for that terminal. This signal normally stops the process. Otherwise, the output generated by that process is output to the current output stream. Processes that are blocking or ignoring **SIGTTOU** signals are excepted and allowed to produce output, if any.

If **FLUSHO** is set, data written to the terminal is discarded. This bit is set when the FLUSH character is typed. A program can cancel the effect of typing the FLUSH character by clearing **FLUSHO**.

If **PENDIN** is set, any input that has not yet been read is reprinted when the next character arrives as input.

If **IEXTEN** is set, the following implementation-defined functions are enabled: special characters (**WERASE, REPRINT, DISCARD**, and **LNEXT**) and local flags (**TOSTOP, ECHOCTL, ECHOPRT, ECHOKE, FLUSHO**, and **PENDIN**).

The initial line-discipline control value is **ISIG, ICANON, ECHO, ECHOK**.

Terminal Size

The number of lines and columns on the terminal's display is specified in the **winsize** structure defined by **sys/termios.h** and includes the following members:

```
unsigned  short   ws_row;    /* rows, in characters */
 unsigned  short   ws_col;    /* columns, in characters */
 unsigned  short   ws_xpixel;/* horizontal size, in pixels */
 unsigned  short   ws_ypixel;/* vertical size, in pixels */
```

termio Structure

The System V **termio** structure is used by some **ioctls**; it is defined by **sys/termio.h** and includes the following members:

```
unsigned  short  c_iflag;    /* input modes */
unsigned  short  c_oflag;    /* output modes */
unsigned  short  c_cflag;    /* control modes */
unsigned  short  c_lflag;    /* local modes */
char             c_line;     /* line discipline */
unsigned  char   c_cc[NCC];  /* control chars */
```

The special control characters are defined by the array **c_cc**. The symbolic name **NCC** is the size of the control-character array and is also defined by **termio.h**. The relative positions, subscript names, and typical default values for each function are as follows:

0	**VINTR**	DEL
1	**VQUIT**	FS
2	**VERASE**	#
3	**VKILL**	@
4	**VEOF**	EOT
5	**VEOL**	NUL
6	**VEOL2**	NUL
7	reserved	

For the non-canonical mode the positions of **VEOF** and **VEOL** are shared by **VMIN** and **VTIME**:

4	**VMIN**	used to set the value of MIN
5	**VTIME**	used to set the value of TIME

The calls that use the **termio** structure only affect the flags and control characters that can be stored in the **termio** structure; all other flags and control characters are unaffected.

Modem lines

On special files representing serial ports, the modem control lines supported by the hardware can be read, and the modem status lines supported by the hardware can be changed. The following modem control and status lines may be supported by a device; they are defined by **sys/termios.h**:

TIOCM_LE	line enable
TIOCM_DTR	data terminal ready
TIOCM_RTS	request to send
TIOCM_ST	secondary transmit
TIOCM_SR	secondary receive
TIOCM_CTS	clear to send
TIOCM_CAR	carrier detect
TIOCM_RNG	ring
TIOCM_DSR	data set ready

TIOCM_CD is a synonym for **TIOCM_CAR**, and **TIOCM_RI** is a synonym for **TIOCM_RNG**. Not all of these are necessarily supported by any particular device; check the manual page for the device in question.

IOCTLS

The **ioctl**s supported by devices and STREAMS modules providing the **termios** interface are listed below. Some calls may not be supported by all devices or modules. The functionality provided by these calls is also available through the preferred function call interface specified on **termios**(2).

TCGETS	The argument is a pointer to a **termios** structure. The current terminal parameters are fetched and stored into that structure.
TCSETS	The argument is a pointer to a **termios** structure. The current terminal parameters are set from the values stored in that structure. The change is immediate.
TCSETSW	The argument is a pointer to a **termios** structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted. This form should be used when changing parameters that affect output.
TCSETSF	The argument is a pointer to a **termios** structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted; all characters queued for input are discarded and then the change occurs.
TCGETA	The argument is a pointer to a **termio** structure. The current terminal parameters are fetched, and those parameters that can be stored in a **termio** structure are stored into that structure.
TCSETA	The argument is a pointer to a **termio** structure. Those terminal parameters that can be stored in a **termio** structure are set from the values stored in that structure. The change is immediate.
TCSETAW	The argument is a pointer to a **termio** structure. Those terminal parameters that can be stored in a **termio** structure are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted. This form should be used when changing parameters that affect output.
TCSETAF	The argument is a pointer to a **termio** structure. Those terminal parameters that can be stored in a **termio** structure are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted; all characters queued for input are discarded and then the change occurs.
TCSBRK	The argument is an **int** value. Wait for the output to drain. If the argument is 0, then send a break (zero valued bits for 0.25 seconds).

TCXONC	Start/stop control. The argument is an **int** value. If the argument is 0, suspend output; if 1, restart suspended output; if 2, suspend input; if 3, restart suspended input.
TCFLSH	The argument is an **int** value. If the argument is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues. On some controllers, if the argument is 0, input flow control characters will be flushed, causing the unflushed output queue to overflow a busy output device.
TIOCGPGRP	The argument is a pointer to a **pid_t**. Set the value of that **pid_t** to the process group ID of the foreground process group associated with the terminal. See **termios**(2) for a description or **TCGETPGRP**.
TIOCSPGRP	The argument is a pointer to a **pid_t**. Associate the process group whose process group ID is specified by the value of that **pid_t** with the terminal. The new process group value must be in the range of valid process group ID values. Otherwise, the error **EPERM** is returned. See **termios**(2) for a description of **TCSETPGRP**.
TIOCGSID	The argument is a pointer to a **pid_t**. The session ID of the terminal is fetched and stored in the **pid_t**.
TIOCGWINSZ	The argument is a pointer to a **winsize** structure. The terminal driver's notion of the terminal size is stored into that structure.
TIOCSWINSZ	The argument is a pointer to a **winsize** structure. The terminal driver's notion of the terminal size is set from the values specified in that structure. If the new sizes are different from the old sizes, a **SIGWINCH** signal is set to the process group of the terminal.
TIOCMBIS	The argument is a pointer to an **int** whose value is a mask containing modem control lines to be turned on. The control lines whose bits are set in the argument are turned on; no other control lines are affected.
TIOCMBIC	The argument is a pointer to an **int** whose value is a mask containing modem control lines to be turned off. The control lines whose bits are set in the argument are turned off; no other control lines are affected.
TIOCMGET	The argument is a pointer to an **int**. The current state of the modem status lines is fetched and stored in the **int** pointed to by the argument.
TIOCMSET	The argument is a pointer to an **int** containing a new set of modem control lines. The modem control lines are turned on or off, depending on whether the bit for that mode is set or clear.

FILES

/dev/*

SEE ALSO

fork(2), **ioctl**(2), **setsid**(2), **signal**(2), **streamio**(7), **termios**(2)

NAME

termiox – extended general terminal interface

DESCRIPTION

The extended general terminal interface supplements the **termio**(7) general terminal interface by adding support for asynchronous hardware flow control, isochronous flow control and clock modes, and local implementations of additional asynchronous features. Some systems may not support all of these capabilities because of either hardware or software limitations. Other systems may not permit certain functions to be disabled. In these cases the appropriate bits will be ignored. See **termiox.h** for your system to find out which capabilities are supported.

Hardware Flow Control Modes

Hardware flow control supplements the **termio**(7) **IXON**, **IXOFF**, and **IXANY** character flow control. Character flow control occurs when one device controls the data transfer of another device by the insertion of control characters in the data stream between devices. Hardware flow control occurs when one device controls the data transfer of another device using electrical control signals on wires (circuits) of the asynchronous interface. Isochronous hardware flow control occurs when one device controls the data transfer of another device by asserting or removing the transmit clock signals of that device. Character flow control and hardware flow control may be simultaneously set.

In asynchronous, full duplex applications, the use of the Electronic Industries Association's EIA-232-D Request To Send (RTS) and Clear To Send (CTS) circuits is the preferred method of hardware flow control. An interface to other hardware flow control methods is included to provide a standard interface to these existing methods.

The EIA-232-D standard specified only uni-directional hardware flow control - the Data Circuit-terminating Equipment or Data Communications Equipment (DCE) indicates to the Data Terminal Equipment (DTE) to stop transmitting data. The **termiox**(7) interface allows both uni-directional and bi-directional hardware flow control; when bi-directional flow control is enabled, either the DCE or DTE can indicate to each other to stop transmitting data across the interface. Note: It is assumed that the asynchronous port is configured as a DTE. If the connected device is also a DTE and not a DCE, then DTE to DTE (for example, terminal or printer connected to computer) hardware flow control is possible by using a null modem to interconnect the appropriate data and control circuits.

Clock Modes

Isochronous communication is a variation of asynchronous communication whereby two communicating devices may provide transmit and/or receive clock to each other. Incoming clock signals can be taken from the baud rate generator on the local isochronous port controller, from CCITT V.24 circuit 114, Transmitter Signal Element Timing - DCE source (EIA-232-D pin 15), or from CCITT V.24 circuit 115, Receiver Signal Element Timing - DCE source (EIA-232-D pin 17). Outgoing clock signals can be sent on CCITT V.24 circuit 113, Transmitter Signal Element Timing - DTE source (EIA-232-D pin 24), on CCITT V.24 circuit 128, Receiver Signal Element Timing - DTE source (no EIA-232-D pin), or not sent at all.

In terms of clock modes, traditional asynchronous communication is implemented simply by using the local baud rate generator as the incoming transmit and receive clock source and not outputting any clock signals.

Terminal Parameters

The parameters that control the behavior of devices providing the **termiox** interface are specified by the **termiox** structure, defined in the **sys/termiox.h** header file. Several **ioctl**(2) system calls that fetch or change these parameters use this structure:

```
#define NFF       5
struct  termiox {
   unsigned short   x_hflag;      /* hardware flow control
                                     modes */
   unsigned short   x_cflag;      /* clock modes */
   unsigned short   x_rflag[NFF];/* reserved modes */
   unsigned short   x_sflag;      /* spare local modes */
};
```

The **x_hflag** field describes hardware flow control modes:

RTSXOFF	0000001	Enable RTS hardware flow control on input.
CTSXON	0000002	Enable CTS hardware flow control on output.
DTRXOFF	0000004	Enable DTR hardware flow control on input.
CDXON	0000010	Enable CD hardware flow control on output.
ISXOFF	0000020	Enable isochronous hardware flow control on input.

The EIA-232-D DTR and CD circuits are used to establish a connection between two systems. The RTS circuit is also used to establish a connection with a modem. Thus, both DTR and RTS are activated when an asynchronous port is opened. If DTR is used for hardware flow control, then RTS must be used for connectivity. If CD is used for hardware flow control, then CTS must be used for connectivity. Thus, RTS and DTR (or CTS and CD) cannot both be used for hardware flow control at the same time. Other mutual exclusions may apply, such as the simultaneous setting of the **termio**(7) **HUPCL** and the **termiox**(7) **DTRXOFF** bits, which use the DTE ready line for different functions.

Variations of different hardware flow control methods may be selected by setting the the appropriate bits. For example, bi-directional RTS/CTS flow control is selected by setting both the **RTSXOFF** and **CTSXON** bits and bi-directional DTR/CTS flow control is selected by setting both the **DTRXOFF** and **CTSXON**. Modem control or uni-directional CTS hardware flow control is selected by setting only the **CTSXON** bit.

As previously mentioned, it is assumed that the local asynchronous port (for example, computer) is configured as a DTE. If the connected device (for example, printer) is also a DTE, it is assumed that the device is connected to the computer's asynchronous port via a null modem that swaps control circuits (typically RTS and CTS). The connected DTE drives RTS and the null modem swaps RTS and CTS so that the remote RTS is received as CTS by the local DTE. In the case that **CTSXON** is set for hardware flow control, printer's lowering of its RTS would cause CTS seen by the computer to be lowered. Output to the printer is suspended until the printer's raising of its RTS, which would cause CTS seen by the computer to be raised.

If **RTSXOFF** is set, the Request To Send (RTS) circuit (line) will be raised, and if the asynchronous port needs to have its input stopped, it will lower the Request To Send (RTS) line. If the RTS line is lowered, it is assumed that the connected device will stop its output until RTS is raised.

If **CTSXON** is set, output will occur only if the Clear To Send (CTS) circuit (line) is raised by the connected device. If the CTS line is lowered by the connected device, output is suspended until CTS is raised.

If **DTRXOFF** is set, the DTE Ready (DTR) circuit (line) will be raised, and if the asynchronous port needs to have its input stopped, it will lower the DTE Ready (DTR) line. If the DTR line is lowered, it is assumed that the connected device will stop its output until DTR is raised.

If **CDXON** is set, output will occur only if the Received Line Signal Detector (CD) circuit (line) is raised by the connected device. If the CD line is lowered by the connected device, output is suspended until CD is raised.

If **ISXOFF** is set, and if the isochronous port needs to have its input stopped, it will stop the outgoing clock signal. It is assumed that the connected device is using this clock signal to create its output. Transit and receive clock sources are programmed using the **x_cflag** fields. If the port is not programmed for external clock generation, **ISXOFF** is ignored. Output isochronous flow control is supported by appropriate clock source programming using the **x_cflag** field and enabled at the remote connected device.

The **x_cflag** field specifies the system treatment of clock modes.

XMTCLK	0000007	Transmit clock source:
XCIBRG	0000000	Get transmit clock from internal baud rate generator.
XCTSET	0000001	Get transmit clock from transmitter signal element timing (DCE source) lead, CCITT V.24 circuit 114, EIA-232-D pin 15.
XCRSET	0000002	Get transmit clock from receiver signal element timing (DCE source) lead, CCITT V.24 circuit 115, EIA-232-D pin 17.
RCVCLK	0000070	Receive clock source:
RCIBRG	0000000	Get receive clock from internal baud rate generator.
RCTSET	0000010	Get receive clock from transmitter signal element timing (DCE source) lead, CCITT V.24 circuit 114, EIA-232-D pin 15.
RCRSET	0000020	Get receive clock from receiver signal element timing (DCE source) lead, CCITT V.24 circuit 115, EIA-232-D pin 17.
TSETCLK	0000700	Transmitter signal element timing (DTE source) lead, CCITT V.24 circuit 113, EIA-232-D pin 24, clock source:
TSETCOFF	0000000	TSET clock not provided.

TSETCRBRG	0000100	Output receive baud rate generator on circuit 113.
TSETCTBRG	0000200	Output transmit baud rate generator on circuit 113.
TSETCTSET	0000300	Output transmitter signal element timing (DCE source) on circuit 113.
TSETCRSET	0000400	Output receiver signal element timing (DCE source) on circuit 113.
RSETCLK	0007000	Receiver signal element timing (DTE source) lead, CCITT V.24 circuit 128, no EIA-232-D pin, clock source:
RSETCOFF	0000000	RSET clock not provided.
RSETCRBRG	0001000	Output receive baud rate generator on circuit 128.
RSETCTBRG	0002000	Output transmit baud rate generator on circuit 128.
RSETCTSET	0003000	Output transmitter signal element timing (DCE source) on circuit 128.
RSETCRSET	0004000	Output receiver signal element timing (DCE) on circuit 128.

If the **XMTCLK** field has a value of **XCIBRG** the transmit clock is taken from the hardware internal baud rate generator, as in normal asynchronous transmission. If **XMTCLK** = **XCTSET** the transmit clock is taken from the Transmitter Signal Element Timing (DCE source) circuit. If **XMTCLK** = **XCRSET** the transmit clock is taken from the Receiver Signal Element Timing (DCE source) circuit.

If the **RCVCLK** field has a value of **RCIBRG** the receive clock is taken from the hardware Internal Baud Rate Generator, as in normal asynchronous transmission. If **RCVCLK** = **RCTSET** the receive clock is taken from the Transmitter Signal Element Timing (DCE source) circuit. If **RCVCLK** = **RCRSET** the receive clock is taken from the Receiver Signal Element Timing (DCE source) circuit.

If the **TSETCLK** field has a value of **TSETCOFF** the Transmitter Signal Element Timing (DTE source) circuit is not driven. If **TSETCLK** = **TSETCRBRG** the Transmitter Signal Element Timing (DTE source) circuit is driven by the Receive Baud Rate Generator. If **TSETCLK** = **TSETCTBRG** the Transmitter Signal Element Timing (DTE source) circuit is driven by the Transmit Baud Rate Generator. If **TSETCLK** = **TSETCTSET** the Transmitter Signal Element Timing (DTE source) circuit is driven by the Transmitter Signal Element Timing (DCE source). If **TSETCLK** = **TSETCRBRG** the Transmitter Signal Element Timing (DTE source) circuit is driven by the Receiver Signal Element Timing (DCE source).

If the **RSETCLK** field has a value of **RSETCOFF** the Receiver Signal Element Timing (DTE source) circuit is not driven. If **RSETCLK** = **RSETCRBRG** the Receiver Signal Element Timing (DTE source) circuit is driven by the Receive Baud Rate Generator. If **RSETCLK** = **RSETCTBRG** the Receiver Signal Element Timing (DTE source) circuit is driven by the Transmit Baud Rate Generator. If **RSETCLK** = **RSETCTSET** the Receiver Signal Element Timing (DTE source) circuit is driven by the Transmitter Signal Element Timing (DCE source). If **RSETCLK** = **RSETCRBRG** the Receiver Signal Element Timing (DTE source) circuit is driven by the Receiver Signal Element Timing (DCE source).

The **x_rflag** is reserved for future interface definitions and should not be used by any implementations. The **x_sflag** may be used by local implementations wishing to customize their terminal interface using the **termiox**(7) ioctl system calls.

ioctls

The **ioctl**(2) system calls have the form:

ioctl (int *fildes***, int** *command***, struct termiox ****arg***);**

The commands using this form are:

TCGETX The argument is a pointer to a **termiox** structure. The current terminal parameters are fetched and stored into that structure.

TCSETX The argument is a pointer to a **termiox** structure. The current terminal parameters are set from the values stored in that structure. The change is immediate.

TCSETXW The argument is a pointer to a **termiox** structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted. This form should be used when changing parameters that will affect output.

TCSETXF The argument is a pointer to a **termiox** structure. The current terminal parameters are set from the values stored in that structure. The change occurs after all characters queued for output have been transmitted; all characters queued for input are discarded and then the change occurs.

FILES

/dev/*

SEE ALSO

ioctl(2), **stty**(1), **termio**(7)

NAME

ticlts, ticots, ticotsord – loopback transport providers

SYNOPSIS

```
#include <ticlts.h>
#include <ticots.h>
#include <ticotsord.h>
```

DESCRIPTION

The devices known as **ticlts**, **ticots**, and **ticotsord** are "loopback transport providers," that is, stand-alone networks at the transport level. Loopback transport providers are transport providers in every sense except one: only one host (the local machine) is "connected to" a loopback network. Loopback transports present a TPI (**STREAMS**-level) interface to application processes and are intended to be accessed via the TLI (application-level) interface. They are implemented as clone devices and support address spaces consisting of "flex-addresses," that is, arbitrary sequences of octets, of length > 0, represented by a **netbuf** structure.

ticlts is a datagram-mode transport provider. It offers (connectionless) service of type **T_CLTS**. Its default address size is **TCL_DEFAULTADDRSZ**. **ticlts** prints the following error messages [see **t_rcvuderr**(3N)]:

TCL_BADADDR	bad address specification
TCL_BADOPT	bad option specification
TCL_NOPEER	bound
TCL_PEERBADSTATE	peer in wrong state

ticots is a virtual circuit-mode transport provider. It offers (connection-oriented) service of type **T_COTS**. Its default address size is **TCO_DEFAULTADDRSZ**. **ticots** prints the following disconnect messages [see **t_rcvdis**(3N)]:

TCO_NOPEER	no listener on destination address
TCO_PEERNOROOMONQ	peer has no room on connect queue
TCO_PEERBADSTATE	peer in wrong state
TCO_PEERINITIATED	peer-initiated disconnect
TCO_PROVIDERINITIATED	provider-initiated disconnect

ticotsord is a virtual circuit-mode transport provider, offering service of type **T_COTS_ORD** (connection-oriented service with orderly release). Its default address size is **TCOO_DEFAULTADDRSZ**. **ticotsord** prints the following disconnect messages [see **t_rcvdis**(3N)]:

TCOO_NOPEER	no listener on destination address
TCOO_PEERNOROOMONQ	peer has no room on connect queue
TCOO_PEERBADSTATE	peer in wrong state
TCOO_PEERINITIATED	peer-initiated disconnect
TCOO_PROVIDERINITIATED	provider-initiated disconnect

USAGE

Loopback transports support a local IPC mechanism through the TLI interface. Applications implemented in a transport provider-independent manner on a client-server model using this IPC are transparently transportable to networked environments.

Transport provider-independent applications must not include the header files listed in the synopsis section above. In particular, the options are (like all transport provider options) provider dependent.

`ticlts` and **`ticots`** support the same service types (**`T_CLTS`** and **`T_COTS`**) supported by the OSI transport-level model. The use of **`ticlts`** and **`ticots is encouraged.`**

`ticotsord` supports the same service type (**`T_COTS_ORD`**) supported by the TCP/IP model. The use of **`ticotsord`** is discouraged except for reasons of compatibility.

FILES

```
/dev/ticlts
/dev/ticots
/dev/ticotsord
```

NAME

timod – Transport Interface cooperating STREAMS module

DESCRIPTION

timod is a STREAMS module for use with the Transport Interface (TI) functions of the Network Services library. The **timod** module converts a set of **ioctl**(2) calls into STREAMS messages that may be consumed by a transport protocol provider which supports the Transport Interface. This allows a user to initiate certain TI functions as atomic operations.

The **timod** module must be pushed onto only a stream terminated by a transport protocol provider which supports the TI.

All STREAMS messages, with the exception of the message types generated from the **ioctl** commands described below, will be transparently passed to the neighboring STREAMS module or driver. The messages generated from the following **ioctl** commands are recognized and processed by the **timod** module. The format of the **ioctl** call is:

```
#include <sys/stropts.h>
          -
          -
struct strioctl strioctl;
          -
          -
strioctl.ic_cmd = cmd;
strioctl.ic_timeout = INFTIM;
strioctl.ic_len = size;
strioctl.ic_dp = (char *)buf
ioctl(fildes, I_STR, &strioctl);
```

Where, on issuance, *size* is the size of the appropriate TI message to be sent to the transport provider and on return *size* is the size of the appropriate TI message from the transport provider in response to the issued TI message. *buf* is a pointer to a buffer large enough to hold the contents of the appropriate TI messages. The TI message types are defined in **sys/tihdr.h**. The possible values for the *cmd* field are:

TI_BIND Bind an address to the underlying transport protocol provider. The message issued to the **TI_BIND ioctl** is equivalent to the TI message type **T_BIND_REQ** and the message returned by the successful completion of the **ioctl** is equivalent to the TI message type **T_BIND_ACK**.

TI_UNBIND Unbind an address from the underlying transport protocol provider. The message issued to the **TI_UNBIND ioctl** is equivalent to the TI message type **T_UNBIND_REQ** and the message returned by the successful completion of the **ioctl** is equivalent to the TI message type **T_OK_ACK**.

TI_GETINFO Get the TI protocol specific information from the transport protocol provider. The message issued to the **TI_GETINFO ioctl** is equivalent to the TI message type **T_INFO_REQ** and the message returned by the successful completion of the **ioctl** is equivalent to the TI message type **T_INFO_ACK**.

TI_OPTMGMT Get, set or negotiate protocol specific options with the transport protocol provider. The message issued to the **TI_OPTMGMT ioctl** is equivalent to the TI message type **T_OPTMGMT_REQ** and the message returned by the successful completion of the **ioctl** is equivalent to the TI message type **T_OPTMGMT_ACK**.

FILES

sys/timod.h
sys/tiuser.h
sys/tihdr.h
sys/errno.h

SEE ALSO

tirdwr(7)

DIAGNOSTICS

If the **ioctl** system call returns with a value greater than 0, the lower 8 bits of the return value will be one of the TI error codes as defined in **sys/tiuser.h**. If the TI error is of type **TSYSERR**, then the next 8 bits of the return value will contain an error as defined in **sys/errno.h** [see **intro**(2)].

NAME

tirdwr – Transport Interface read/write interface STREAMS module

DESCRIPTION

tirdwr is a STREAMS module that provides an alternate interface to a transport provider which supports the Transport Interface (TI) functions of the Network Services library (see Section 3N). This alternate interface allows a user to communicate with the transport protocol provider using the **read**(2) and **write**(2) system calls. The **putmsg**(2) and **getmsg**(2) system calls may also be used. However, **putmsg** and **getmsg** can only transfer data messages between user and stream.

The **tirdwr** module must only be pushed [see **I_PUSH** in **streamio**(7)] onto a stream terminated by a transport protocol provider which supports the TI. After the **tirdwr** module has been pushed onto a stream, none of the Transport Interface functions can be used. Subsequent calls to TI functions will cause an error on the stream. Once the error is detected, subsequent system calls on the stream will return an error with **errno** set to **EPROTO**.

The following are the actions taken by the **tirdwr** module when pushed on the stream, popped [see **I_POP** in **streamio**(7)] off the stream, or when data passes through it.

push When the module is pushed onto a stream, it will check any existing data destined for the user to ensure that only regular data messages are present. It will ignore any messages on the stream that relate to process management, such as messages that generate signals to the user processes associated with the stream. If any other messages are present, the **I_PUSH** will return an error with **errno** set to **EPROTO**.

write The module will take the following actions on data that originated from a **write** system call:

> All messages with the exception of messages that contain control portions (see the **putmsg** and **getmsg** system calls) will be transparently passed onto the module's downstream neighbor.
>
> Any zero length data messages will be freed by the module and they will not be passed onto the module's downstream neighbor.
>
> Any messages with control portions will generate an error, and any further system calls associated with the stream will fail with **errno** set to **EPROTO**.

read The module will take the following actions on data that originated from the transport protocol provider:

> All messages with the exception of those that contain control portions (see the **putmsg** and **getmsg** system calls) will be transparently passed onto the module's upstream neighbor.
>
> The action taken on messages with control portions will be as follows:

Messages that represent expedited data will generate an error. All further system calls associated with the stream will fail with **errno** set to **EPROTO**.

Any data messages with control portions will have the control portions removed from the message prior to passing the message on to the upstream neighbor.

Messages that represent an orderly release indication from the transport provider will generate a zero length data message, indicating the end of file, which will be sent to the reader of the stream. The orderly release message itself will be freed by the module.

Messages that represent an abortive disconnect indication from the transport provider will cause all further **write** and **putmsg** system calls to fail with **errno** set to **ENXIO**. All further **read** and **getmsg** system calls will return zero length data (indicating end of file) once all previous data has been read.

With the exception of the above rules, all other messages with control portions will generate an error and all further system calls associated with the stream will fail with **errno** set to **EPROTO**.

Any zero length data messages will be freed by the module and they will not be passed onto the module's upstream neighbor.

pop When the module is popped off the stream or the stream is closed, the module will take the following action:

If an orderly release indication has been previously received, then an orderly release request will be sent to the remote side of the transport connection.

SEE ALSO

getmsg(2), **intro**(2), **intro**(3) **putmsg**(2), **read**(2), **streamio**(7), **timod**(7), **write**(2)

NAME

ttcompat – V7, 4BSD and XENIX STREAMS compatibility module

SYNOPSIS

```
#include <sys/stream.h>
#include <sys/stropts.h>
#include <sys/ttcompat.h>
#include <sys/ttold.h>

ioctl(fd, I_PUSH, "ttcompat");
```

DESCRIPTION

ttcompat is a STREAMS module that translates the **ioctl** calls supported by the older Version 7, 4BSD and XENIX terminal drivers into the **ioctl** calls supported by the **termio** interface [see **termio**(7)]. All other messages pass through this module unchanged; the behavior of **read** and **write** calls is unchanged, as is the behavior of **ioctl** calls other than the ones supported by **ttcompat**.

This module can be automatically pushed onto a stream with the **autopush**(1M) mechanism when a terminal device is opened; it does not have to be explicitly pushed onto a stream. This module requires that the **termios** interface be supported by the modules and the application can push the driver downstream. The **TCGETS**, **TCSETS**, and **TCSETSF** **ioctl** calls must be supported; if any information set or fetched by those **ioctl** calls is not supported by the modules and driver downstream, some of the V7/4BSD/XENIX functions may not be supported. For example, if the **CBAUD** bits in the **c_cflag** field are not supported, the functions provided by the **sg_ispeed** and **sg_ospeed** fields of the **sgttyb** structure (see below) will not be supported. If the **TCFLSH** **ioctl** is not supported, the function provided by the **TIOCFLUSH** **ioctl** will not be supported. If the **TCXONC** **ioctl** is not supported, the functions provided by the **TIOCSTOP** and **TIOCSTART** **ioctl** calls will not be supported. If the **TIOCMBIS** and **TIOCMBIC** **ioctl** calls are not supported, the functions provided by the **TIOCSDTR** and **TIOCCDTR** **ioctl** calls will not be supported.

The basic **ioctl** calls use the **sgttyb** structure defined by **sys/ioctl.h**:

```
struct sgttyb {
        char    sg_ispeed;
        char    sg_ospeed;
        char    sg_erase;
        char    sg_kill;
        int     sg_flags;
};
```

The **sg_ispeed** and **sg_ospeed** fields describe the input and output speeds of the device, and reflect the values in the **c_cflag** field of the **termios** structure. The **sg_erase** and **sg_kill** fields of the argument structure specify the erase and kill characters respectively, and reflect the values in the **VERASE** and **VKILL** members of the **c_cc** field of the **termios** structure.

The **sg_flags** field of the argument structure contains several flags that determine the system's treatment of the terminal. They are mapped into flags in fields of the terminal state, represented by the **termios** structure.

Delay type **0** is always mapped into the equivalent delay type **0** in the **c_oflag** field of the **termios** structure. Other delay mappings are performed as follows:

sg_flags	c_oflag
BS1	BS1
FF1	VT1
CR1	CR2
CR2	CR3
CR3	not supported
TAB1	TAB1
TAB2	TAB2
XTABS	TAB3
NL1	ONLRET|CR1
NL2	NL1

If previous **TIOCLSET** or **TIOCLBIS ioctl** calls have not selected **LITOUT** or **PASS8** mode, and if **RAW** mode is not selected, the **ISTRIP** flag is set in the **c_iflag** field of the **termios** structure, and the **EVENP** and **ODDP** flags control the parity of characters sent to the terminal and accepted from the terminal:

Parity is not to be generated on output or checked on input:

The character size is set to **CS8** and the flag is cleared in the **c_cflag** field of the **termios** structure.

Even parity characters are to be generated on output and accepted on input:

The flag is set in the **c_iflag** field of the **termios** structure, the character size is set to **CS7** and the flag is set in the **c_cflag** field of the **termios** structure.

Odd parity characters are to be generated on output and accepted on input:

The flag is set in the **c_iflag** field, the character size is set to **CS7** and the and flags are set in the **c_cflag** field of the **termios** structure.

Even parity characters are to be generated on output and characters of either parity are to be accepted on input:

The flag is cleared in the **c_iflag** field, the character size is set to **CS7** and the flag is set in the **c_cflag** field of the **termios** structure.

The **RAW** flag disables all output processing (the **OPOST** flag in the **c_oflag** field, and the **XCASE** flag in the **c_lflag** field, are cleared in the **termios** structure) and input processing (all flags in the **c_iflag** field other than the **IXOFF** and **IXANY** flags are cleared in the **termios** structure). 8 bits of data, with no parity bit, are accepted on input and generated on output; the character size is set to **CS8** and the **PARENB** and **PARODD** flags are cleared in the **c_cflag** field of the **termios** structure. The signal-generating and line-editing control characters are disabled by clearing the **ISIG** and **ICANON** flags in the **c_lflag** field of the **termios** structure.

The **CRMOD** flag turns input RETURN characters into NEWLINE characters, and output and echoed NEWLINE characters to be output as a RETURN followed by a LINEFEED. The **ICRNL** flag in the **c_iflag** field, and the **OPOST** and **ONLCR** flags in the **c_oflag** field, are set in the **termios** structure.

The **LCASE** flag maps upper-case letters in the ASCII character set to their lower-case equivalents on input (the **IUCLC** flag is set in the **c_iflag** field), and maps lower-case letters in the ASCII character set to their upper-case equivalents on output (the **OLCUC** flag is set in the **c_oflag** field). Escape sequences are accepted on input, and generated on output, to handle certain ASCII characters not supported by older terminals (the **XCASE** flag is set in the **c_lflag** field).

Other flags are directly mapped to flags in the **termios** structure:

sg_flags	flags in **termios** structure
CBREAK	complement of **ICANON** in **c_lflag** field
ECHO	**ECHO** in **c_lflag** field
TANDEM	**IXOFF** in **c_iflag** field

Another structure associated with each terminal specifies characters that are special in both the old Version 7 and the newer 4BSD terminal interfaces. The following structure is defined by **sys/ioctl.h**:

```
struct tchars {
        char   t_intrc;       /* interrupt */
        char   t_quitc;       /* quit */
        char   t_startc;      /* start output */
        char   t_stopc;       /* stop output */
        char   t_eofc;        /* end-of-file */
        char   t_brkc;        /* input delimiter (like nl) */
};
```

XENIX defines the **tchar** structure as **tc**. The characters are mapped to members of the **c_cc** field of the **termios** structure as follows:

tchars	**c_cc** index
t_intrc	**VINTR**
t_quitc	**VQUIT**
t_startc	**VSTART**
t_stopc	**VSTOP**
t_eofc	**VEOF**
t_brkc	**VEOL**

Also associated with each terminal is a local flag word, specifying flags supported by the new 4BSD terminal interface. Most of these flags are directly mapped to flags in the **termios** structure:

local flags	flags in **termios** structure
LCRTBS	not supported
LPRTERA	**ECHOPRT** in the **c_lflag** field
LCRTERA	**ECHOE** in the **c_lflag** field
LTILDE	not supported
LTOSTOP	**TOSTOP** in the **c_lflag** field
LFLUSHO	**FLUSHO** in the **c_lflag** field
LNOHANG	**CLOCAL** in the **c_cflag** field

LCRTKIL	ECHOKE in the c_lflag field
LCTLECH	CTLECH in the c_lflag field
LPENDIN	PENDIN in the c_lflag field
LDECCTQ	complement of IXANY in the c_iflag field
LNOFLSH	NOFLSH in the c_lflag field

Another structure associated with each terminal is the **ltchars** structure which defines control characters for the new 4BSD terminal interface. Its structure is:

```
struct ltchars {
    char  t_suspc;      /* stop process signal */
    char  t_dsuspc;     /* delayed stop process signal */
    char  t_rprntc;     /* reprint line */
    char  t_flushc;     /* flush output (toggles) */
    char  t_werasc;     /* word erase */
    char  t_lnextc;     /* literal next character */
};
```

The characters are mapped to members of the **c_cc** field of the **termios** structure as follows:

ltchars	c_cc index
t_suspc	VSUSP
t_dsuspc	VDSUSP
t_rprntc	VREPRINT
t_flushc	VDISCARD
t_werasc	VWERASE
t_lnextc	VLNEXT

ioctls

ttcompat responds to the following **ioctl** calls. All others are passed to the module below.

TIOCGETP The argument is a pointer to an **sgttyb** structure. The current terminal state is fetched; the appropriate characters in the terminal state are stored in that structure, as are the input and output speeds. The values of the flags in the **sg_flags** field are derived from the flags in the terminal state and stored in the structure.

TIOCEXCL Set "exclusive-use" mode; no further opens (except by a privileged user) are permitted until the file has been closed.

TIOCNXCL Turn off "exclusive-use" mode.

TIOCSETP The argument is a pointer to an **sgttyb** structure. The appropriate characters and input and output speeds in the terminal state are set from the values in that structure, and the flags in the terminal state are set to match the values of the flags in the **sg_flags** field of that structure. The state is changed with a **TCSETSF ioctl** so that the interface delays until output is quiescent, then throws away any unread characters, before changing the modes.

TIOCSETN	The argument is a pointer to an **sgttyb** structure. The terminal state is changed as **TIOCSETP** would change it, but a **TCSETS ioctl** is used, so that the interface neither delays nor discards input.
TIOCHPCL	The argument is ignored. The **HUPCL** flag is set in the **c_cflag** word of the terminal state.
TIOCFLUSH	The argument is a pointer to an **int** variable. If its value is zero, all characters waiting in input or output queues are flushed. Otherwise, the value of the **int** is treated as the logical **OR** of the **FREAD** and **FWRITE** flags defined by **sys/file.h**; if the **FREAD** bit is set, all characters waiting in input queues are flushed, and if the **FWRITE** bit is set, all characters waiting in output queues are flushed.
TIOCBRK	The argument is ignored. The break bit is set for the device.
TIOCCBRK	The argument is ignored. The break bit is cleared for the device.
TIOCSDTR	The argument is ignored. The Data Terminal Ready bit is set for the device.
TIOCCDTR	The argument is ignored. The Data Terminal Ready bit is cleared for the device.
TIOCSTOP	The argument is ignored. Output is stopped as if the **STOP** character had been typed.
TIOCSTART	The argument is ignored. Output is restarted as if the **START** character had been typed.
TIOCGETC	The argument is a pointer to a **tchars** structure. The current terminal state is fetched, and the appropriate characters in the terminal state are stored in that structure.
TIOCSETC	The argument is a pointer to a **tchars** structure. The values of the appropriate characters in the terminal state are set from the characters in that structure.
TIOCLGET	The argument is a pointer to an **int**. The current terminal state is fetched, and the values of the local flags are derived from the flags in the terminal state and stored in the **int** pointed to by the argument.
TIOCLBIS	The argument is a pointer to an **int** whose value is a mask containing flags to be set in the local flags word. The current terminal state is fetched, and the values of the local flags are derived from the flags in the terminal state; the specified flags are set, and the flags in the terminal state are set to match the new value of the local flags word.
TIOCLBIC	The argument is a pointer to an **int** whose value is a mask containing flags to be cleared in the local flags word. The current terminal state is fetched, and the values of the local flags are derived from the flags in the terminal state; the specified flags are cleared, and the flags in the terminal state are set to match the new value of the local flags word.

`TIOCLSET`	The argument is a pointer to an `int` containing a new set of local flags. The flags in the terminal state are set to match the new value of the local flags word.
`TIOCGLTC`	The argument is a pointer to an `ltchars` structure. The values of the appropriate characters in the terminal state are stored in that structure.
`TIOCSLTC`	The argument is a pointer to an `ltchars` structure. The values of the appropriate characters in the terminal state are set from the characters in that structure.
`FIORDCHK`	`FIORDCHK` returns the number of immediately readable characters. The argument is ignored.
`FIONREAD`	`FIONREAD` returns the number of immediately readable characters in the `int` pointed to by the argument.
`LDSMAP`	Calls the function **emsetmap**(*tp*, *mp*) if the function is configured in the kernel.
`LDGMAP`	Calls the function **emgetmap**(*tp*, *mp*) if the function is configured in the kernel.
`LDNMAP`	Calls the function **emunmap**(*tp*, *mp*) if the function is configured in the kernel.

The following **ioctls** are returned as successful for the sake of compatibility. However, nothing significant is done (that is, the state of the terminal is not changed in any way).

```
TIOCSETD   LDOPEN
TIOCGETD   LDCLOSE
DIOCSETP   LDCHG
DIOCSETP   LDSETT
DIIOGETP   LDGETT
```

SEE ALSO

ioctl(2), **ldterm**(7), **termio**(7), **termios**(2)

NOTES

TIOCBRK and **TIOCCBRK** should be handled by the driver. **FIONREAD** and **FIORDCHK** are handled in the stream head.

NAME

`tty` – controlling terminal interface

DESCRIPTION

The file **`/dev/tty`** is, in each process, a synonym for the control terminal associated with the process group of that process, if any. It is useful for programs or shell sequences that wish to be sure of writing messages on the terminal no matter how output has been redirected. It can also be used for programs that demand the name of a file for output, when typed output is desired and it is tiresome to find out what terminal is currently in use.

FILES

`/dev/tty`
`/dev/tty*`

SEE ALSO

`console`(7)

NAME

UDP – Internet User Datagram Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_DGRAM, 0);

t = t_open("/dev/udp", O_RDWR);
```

DESCRIPTION

UDP is a simple datagram protocol which is layered directly above the Internet Protocol (IP). Programs may access UDP using the socket interface, where it supports the **SOCK_DGRAM** socket type, or using the Transport Level Interface (TLI), where it supports the connectionless (**T_CLTS**) service type.

Within the socket interface, UDP is normally used with the **sendto()**, **sendmsg()**, **recvfrom()**, and **recvmsg()** calls [see **send**(3N) and **recv**(3N)]. If the **connect**(3N) call is used to fix the destination for future packets, then the **recv**(3N) or **read**(2) and **send**(3N) or **write**(2) calls may be used.

UDP address formats are identical to those used by the Transmission Control Protocol (TCP). Like TCP, UDP uses a port number along with an IP address to identify the endpoint of communication. The UDP port number space is separate from the TCP port number space (that is, a UDP port may not be connected to a TCP port). The **bind**(3N) call can be used to set the local address and port number of a UDP socket. The local IP address may be left unspecified in the **bind()** call by using the special value **INADDR_ANY**. If the **bind()** call is not done, a local IP address and port number will be assigned to the endpoint when the first packet is sent. Broadcast packets may be sent (assuming the underlying network supports this) by using a reserved broadcast address. This address is network interface dependent. Broadcasts may only be sent by the privileged user.

Options at the IP level may be used with UDP; see **ip**(7).

As the RFC allows, there are a variety of ways that a UDP packet can be lost or corrupted, including a failure of the underlying communication mechanism. UDP implements a checksum over the data portion of the packet. If the checksum of a received packet is in error, the packet will be dropped with no indication given to the user. A queue of received packets is provided for each UDP socket. This queue has a limited capacity. Arriving datagrams which will not fit within its *high-water* capacity are silently discarded.

As the RFC allows, UDP processes Internet Control Message Protocol (ICMP) error messages received in response to UDP packets it has sent. See **icmp**(7). ICMP source quench messages are ignored. ICMP destination unreachable, time exceeded and parameter problem messages disconnect the socket from its peer so that subsequent attempts to send packets using that socket will return an error. UDP will not guarantee that packets are delivered in the order they were sent. As well, duplicate packets may be generated in the communication process.

SEE ALSO

bind(3N), **connect**(3N), **icmp**(7), **inet**(7), **ip**(7), **read**(2), **recv**(3N), **send**(3N), **tcp**(7), **write**(2)

Postel, Jon, *User Datagram Protocol*, RFC 768, Network Information Center, SRI International, Menlo Park, Calif., August 1980

DIAGNOSTICS

A socket operation may fail if:

EISCONN	A **connect()** operation was attempted on a socket on which a **connect()** operation had already been performed, and the socket could not be successfully disconnected before making the new connection.
EISCONN	A **sendto()** or **sendmsg()** operation specifying an address to which the message should be sent was attempted on a socket on which a **connect()** operation had already been performed.
ENOTCONN	A **send()** or **write()** operation, or a **sendto()** or **sendmsg()** operation not specifying an address to which the message should be sent, was attempted on a socket on which a **connect()** operation had not already been performed.
EADDRINUSE	A **bind()** operation was attempted on a socket with a network address/port pair that has already been bound to another socket.
EADDRNOTAVAIL	A **bind()** operation was attempted on a socket with a network address for which no network interface exists.
EINVAL	A **sendmsg()** operation with a non-NULL **msg_accrights** was attempted.
EACCES	A **bind()** operation was attempted with a reserved port number and the effective user ID of the process was not the privileged user.
ENOBUFS	The system ran out of memory for internal data structures.

NAME

vxfsio – **vxfs** file system control functions

SYNOPSIS

#include <**sys/types.h**>
#include <**sys/fs/vx_ioctl.h**>

int ioctl (int *fildes*, **int** *cmd*, *arg*)**;**

DESCRIPTION

The **vxfs ioctl**(2) enhancements provide for extended control over open files.

The argument *fildes* is an open file_descriptor.

The data type and value of *arg* are specific to the type of command specified by *cmd*. Unless specified, *arg* is treated as an **int** type. The symbolic names for commands and file status flags are defined by the **sys/fs/vx_ioctl.h** header file.

The enhancements available are:

VX_SETCACHE

Set caching advisories. These advisories allow an application to indicate to the file system which forms of caching would be most advantageous.

NOTE: **VX_SETCACHE** is available with the VxFS Advanced package only.

The values for *arg* are such that multiple advisories may be set by combining values with bitwise **OR** operations. The possible values for *arg* are:

VX_RANDOM

Indicates that the file is being accessed randomly. Read-ahead should not be performed.

VX_SEQ

Indicates that the file is being accessed sequentially. Maximum read-ahead should be performed.

VX_DIRECT

Indicates that data associated with read and write operations is to be transferred directly to or from the user supplied buffer, without being cached. When this options is enabled, all I/O operations must begin on block boundaries and must be a multiple of the block size in length. The buffer supplied with the I/O operations must be aligned to a page boundary.

If an I/O request fails to meet alignment criteria, or the file is currently being accessed for mapped I/O, the I/O request will be performed as a data synchronous I/O operation.

VX_NOREUSE

Indicates that buffered data does not need to be retained in anticipation of further use by the application.

VX_DSYNC

Indicates that data synchronous I/O mode is desired. In data synchronous I/O mode, a write operation returns to the caller after the data has been transferred to external media, but the inode is not updated synchronously if only the times in the inode need to be updated.

Only one of **VX_RANDOM**, **VX_SEQ**, or **VX_DIRECT** may be specified. The **VX_NOREUSE** and **VX_DSYNC** options may not be used in conjunction with **VX_DIRECT**. The caching advisories for a file are maintained on a per-file basis. Changes made to the advisories by a process affect I/O operations by all processes currently accessing the file.

The **VX_SETCACHE** ioctl returns a 0 if the caching advisories are successfully set. If the operation fails, the return value is -1 and the external variable **errno** will be a general DIAGNOSTIC.

VX_GETCACHE

Get caching advisories in effect for the file. The argument *arg* should be a pointer to to an **int**.

The **VX_GETCACHE** ioctl returns a 0 if the caching advisories are successfully obtained and the advisories are returned in *arg*. If the operation fails, the return value is -1 and the external variable **errno** will be a general DIAGNOSTIC.

VX_SETEXT

Set extent information.

NOTE: **VX_SETEXT** is available with the VxFS Advanced package only.

The extent information is set according to the parameters specified by *arg*. The argument *arg* points to a structure of type **vx_ext** defined in **sys/fs/vx_ioctl.h**, which contains the following members:

```
off_t    ext_size; /* extent size */
off_t    reserve;  /* space reservation */
int      a_flags;  /* allocation flags */
```

The **ext_size** element is used to request a fixed extent size, in blocks, for the file. If a fixed extent size is not required, zero should be used to allow the default allocation policy to be used. Changes to the fixed extent size made after the file contains indirect blocks have no effect unless all current indirect blocks are freed via file truncation and/or reservation deallocation.

The **reserve** element is used to set the amount of space preallocated to the file. If the **reserve** amount is greater than the current reservation, the allocation for the file is increased to match the **reserve** amount. If the **reserve** amount is less than the current reservation, the allocation is decreased. The allocation will not be reduced to less than the current file size.

File reservation cannot be increased beyond the **ulimit(2)** of the requesting process. However, an existing reservation will not be trimmed to the requesting process's **ulimit(2)**. Reservation of space for existing sparse files will not cause blocks to be allocated to fill in the holes, but will only allocate blocks after the end of the file.

Thus, it's possible to have a larger reservation for a file than blocks in the file.

The reservation amount is independent of file size since reservation is used to preallocate space for a file. The **a_flags** element is used to indicate the type of reservation required. The choices are:

VX_NOEXTEND
: The file may not be extended once the current reservation is exceeded. The reservation may be increased if necessary by another invocation of the ioctl, but the file will not be automatically extended.

VX_TRIM
: The reservation for the file is to be trimmed to the current file size upon last close by all processes that have the file open.

VX_CONTIGUOUS
: The reservation must be allocated contiguously (as a single extent). **ext_size** will become the fixed extent size for subsequent allocations, but has no affect on this one. The reservation will fail if the file has gone into indirect extents, unless the amount of space requested is the same as the indirect extent size. If the contiguous allocation request is done on an empty file, this will not happen.

VX_ALIGN
: Align all new extents on an **ext_size** boundary relative to the starting block of an allocation unit. If **VX_CONTIGUOUS** is set, the single extent allocated during this invocation is not subject to the alignment restriction.

VX_NORESERVE
: The reservation is to be made as a non-persistent allocation to the file. The on-disk inode will not be updated with the reservation information so that the reservation will not survive a system crash. The reservation is associated with the file until the close of the file. The reservation is trimmed to the current file size on close.

VX_CHGSIZE
: The reservation is to be immediately incorporated into the file. The file's on-disk inode is updated with the size and block count information that is increased to include the reserved space. Unlike an **fcntl F_FREESP** operation which "truncates-up" [see **fcntl**(2)], the space included in the file is not initialized. This operation is restricted to users with appropriate privileges.

Write permission to a file is required to set extent information, but any process that can open the file can get the extent information. Extent information only applies to regular files. Only one set of extent information is kept per file. Except in those cases noted above as non-persistent, the extent information becomes part of the on-disk inode information, and thus persists across reboots.

The **VX_SETEXT** ioctl returns a 0 if the extent information is successfully set. If the operation fails, the return value is -1 and the external variable **errno** will be a general DIAGNOSTIC.

VX_GETEXT

Get extent information. Return the extent information associated with *fildes*. The argument *arg* points to a structure of type **vx_ext** as defined in **sys/fs/vx_ioctl.h**.

The **VX_GETEXT** ioctl returns a 0 if the extent information is successfully obtained. If the operation fails, the return value is -1 and the external variable **errno** will be a general DIAGNOSTIC.

VX_GETFSOPT

Get file system options. The argument *arg* should be a pointer to to an **int**. This command may be used by any user who can open the root inode on the file system. The options returned in *arg* are:

VX_FSO_BLKCLEAR
: Indicates that all newly allocated blocks will be guaranteed to contain all zeros.

VX_FSO_CACHE_CLOSESYNC
: Indicates that any non-logged changes to the inode or data will be flushed to disk when the file is closed.

VX_FSO_CACHE_DIRECT
: Indicates that any non-synchronous I/O will be handled as if the **VX_DIRECT** cache advisory had been set on the file. Also, any non-logged changes to the inode or data will be flushed to disk when the file is closed.

VX_FSO_CACHE_DSYNC
: Indicates that any writes that don't have either **O_SYNC** or the **VX_DIRECT** advisory set will be handled as if the **VX_DSYNC** advisory had been set on the file. Also, any non-logged changes to the inode or data will be flushed to disk when the file is closed.

VX_FSO_NODATAINLOG
: Indicates that intent logging of user data for synchronous writes is disabled.

VX_FSO_NOLOG
: Indicates that intent logging of structural changes to the file system is disabled.

VX_FSO_OSYNC_CLOSESYNC
: Indicates that any non-logged changes to the inode or data will be flushed to disk when a file accessed with **O_SYNC** is closed.

VX_FSO_OSYNC_DIRECT
Indicates that any **O_SYNC** I/O will be handled as if the **VX_DIRECT** cache advisory had been set on the file instead. Also, any non-logged changes to the inode or data will be flushed to disk when a file accessed with **O_SYNC** is closed.

VX_FSO_OSYNC_DSYNC
Indicates that any **O_SYNC** writes will be handled as if the **VX_DSYNC** cache advisory had been set on the file instead. Also, any non-logged changes to the inode or data will be flushed to disk when a file accessed with **O_SYNC** is closed.

VX_FSO_SNAPPED
Indicates that a snapshot backup is in progress on the file system.

VX_FSO_SNAPSHOT
Indicates that this file system is a snapshot backup of another file system.

VX_FSO_VJFS
Indicates that this is not the VxFS Advanced package.

The **VX_GETFSOPT** ioctl returns a 0 if the file system options are successfully obtained. If the operation fails, the return value is -1 and the external variable **errno** will be a general DIAGNOSTIC.

VX_FREEZE
Sync then freeze the file system. Once frozen, all further operations against the file system block until a **VX_THAW** operation is received. The argument *arg* is a timeout value expressed in seconds. If a **VX_THAW** operation is not received within the specified timeout interval, the file system will perform a **VX_THAW** operation automatically.

This command may only be used by a user with appropriate privilege, on the root directory of the file system.

The **VX_FREEZE** ioctl returns a 0 if the file system is successfully frozen. If the operation fails, the return value is -1 and the external variable **errno** will be a general DIAGNOSTIC.

VX_THAW
Unblock a file system that has been frozen by a **VX_FREEZE** operation. The argument *arg* should be NULL. The process that is to issue a **VX_THAW** operation must have the root directory of the file system open, and must ensure that it does not access the file system after the file system has been frozen, to ensure that the process itself does not block.

This command may only be used by a user with appropriate privilege, on the root directory of the file system.

The **VX_THAW** ioctl returns a 0 if the file system is successfully unfrozen. If the operation fails, the return value is -1 and the external variable **errno** will be a general DIAGNOSTIC or one of the following:

DIAGNOSTICS

The following values are returned in **errno** upon operation failures:

EACCESS	The calling process does not have write access to the file specified by *fildes*.
EAGAIN	The file system is not currently frozen.
EFBIG	An attempt was made to reserve space larger than the maximum file size limit for this process.
EINVAL	The command or argument is invalid.
EPERM	The process does not have appropriate privilege.
ENODEV	The file specified by *fildes* is not the root directory of a **vxfs** file system.
EROFS	The file system is mounted read-only.
EIO	An I/O error occurred while attempting to perform the operation.
ENOSPC	Requested space could not be obtained.
EFAULT	An address specified by an argument is invalid.

SEE ALSO

getrlimit (2), ioctl (2), ulimit (2)

NAME

wd – Western Digital WD8003 Ethernet Driver

SYNOPSIS

```
#include <sys/dlpi.h>
#include <sys/dlpi_ether.h>
#include <sys/wd.h>

fd = open ("/dev/wd_0", O_RDWR)
```

DESCRIPTION

The **wd** driver provides a data link interface to the WD8003 family of ISA and MCA Ethernet controllers from Western Digital. It is a STREAMS-based driver, compatible with the Data Link Provider Interface (DLPI) and Logical Link Interface (LLI) software interfaces.

The **wd** driver supports both **DL_ETHER** and **DL_CSMACD** for MAC type, **DL_CL_ETHER** for service mode, and **DL_STYLE1** for provider style. The driver can operate as a cloned or non-cloned device.

A process must issue a **DL_BIND_REQ** primitive to receive frames from the network. This primitive includes a **dl_bind_req_t** structure.

The process must specify the *dl_sap* field of the **dl_bind_req_t** structure in host order. The type field of an incoming frame is converted to host order and compared to the *dl_sap* value. If the values are equal, the frame is placed on the STREAMS read queue of the requesting process. A privileged process may set the *dl_sap* field to **PROMISCUOUS_SAP**. The **PROMISCUOUS_SAP** matches all incoming frames.

A privileged process may bind to an SAP already bound by another process. In cases where a frame qualifies to be sent to more than one process, independent copies of the frame are made and placed on the STREAMS read queue of each process.

Received frames are delivered in **dl_unitdata_ind_t** structures. The source and destination address each contain a 6-byte Ethernet address, followed by a 2-byte type value, written in network order.

ioctl Calls

The following **ioctl**s are supported:

DLIOCGMIB

Returns the **DL_mib_t** structure, which contains the Management Information Base (MIB). The MIB holds the Ethernet statistics kept in the driver.

```
/*
 *  Ether statistics structure.
 */
typedef struct {
  ulong_tetherAlignErrors;    /* Frame alignment errors */
  ulong_tetherCRCerrors;      /* CRC errors */
  ulong_tetherMissedPkts;     /* Packet overflow or missed inter */
  ulong_tetherOverrunErrors;  /* Overrun errors */
  ulong_tetherUnderrunErrors; /* Underrun errors */
  ulong_tetherCollisions;     /* Total collisions */
 ulong_t etherAbortErrors;    /* Transmits aborted at interface */
  ulong_tetherCarrierLost;    /* Carrier sense signal lost */
  ulong_tetherReadqFull;      /* STREAMS read queue full */
  ulong_tetherRcvResources;   /* Receive resource alloc faliure
*/
  ulong_tetherDependent1;     /* Device dependent statistic */
  ulong_tetherDependent2;     /* Device dependent statistic */
  ulong_tetherDependent3;     /* Device dependent statistic */
  ulong_tetherDependent4;     /* Device dependent statistic */
  ulong_tetherDependent5;     /* Device dependent statistic */
} DL_etherstat_t;

/*
 *  Interface statistics compatible with MIB II SNMP requirements.
 */
typedef struct {
  int     ifIndex;            /* 1 through ifNumber */
  int     ifDescrLen;         /* len of desc. following this struct */
  int     ifType;             /* type of interface */
  int     ifMtu;              /* datagram size that can be sent/rcv */
  ulong_tifSpeed;             /* estimate of bandwith in bits PS */
  uchar_tifPhyAddress[DL_MAC_ADDR_LEN];/* Ethernet Address */
  int     ifAdminStatus;      /* desired state of the interface */
  int     ifOperStatus;       /* current state of the interface */
  ulong_tifLastChange;        /* sysUpTime when state was entered */
  ulong_tifInOctets;          /* octets received on interface */
  ulong_tifInUcastPkts;       /* unicast packets delivered */
  ulong_tifInNUcastPkts;      /* non-unicast packets delivered */
  ulong_tifInDiscards;        /* good packets received but dropped */
  ulong_tifInErrors;          /* packets received with errors */
  ulong_tifInUnknownProtos;   /* packets recv'd to unbound proto
*/
  ulong_tifOutOctets;         /* octets transmitted on interface '*/
  ulong_tifOutUcastPkts;      /* unicast packets transmited */
  ulong_tifOutNUcastPkts;     /* non-unicast packets transmited */
  ulong_tifOutDiscards;       /* good outbound packets dropped */
  ulong_tifOutErrors;         /* number of transmit errors */
  ulong_tifOutQlen;           /* length of output queue */
  DL_etherstat_t ifSpecific;  /* Ethernet specific stats */
} DL_mib_t;
```

The values in the MIB are compatible with those needed by the SNMP protocol.

The *ifDescrLen* field indicates the length of the null-terminated description string that immediately follows the `DL_mib_t` structure.

There are three fields in the MIB that are specific to the **wd** driver: The *ifSpecific.etherDependent1* field tracks the number of times the transceiver failed to transmit a collision signal after transmission of a packet. The *ifSpecific.etherDependent2* field tracks the number of collisions that occurred after a slot time (out-of-window collisions). The *ifSpecific.etherDependent3* field tracks the number of times a transmit interrupt timeout condition occurred.

DLIOCSMIG
Allows a privileged process to initialize the values in the MIB (that is, the **DL_mib_t** structure). A process cannot use this **ioctl** to change the **ifPhyAddress**, the **ifDescrLen**, or the text of the description fields.

DLIOCGENADDR
Returns the Ethernet address in network order.

DLIOCGLPCFLG
Returns the state of the local packet copy flag in the *ioc_rval* field of the **iocblk** structure. The local copy flag determines if packets looped back by the driver should also be sent to the network. A non-zero value indicates that frames should also be be sent to the network after being looped back. The default value of this flag is zero.

DLIOCSLPCFLG
Allows a privileged process to set the local packet copy flag, causing all packets looped back by the driver to also be sent to the network.

DLIOCGPROMISC
Returns the value of the promiscuous flag in the *ioc_rval* of the **iocblk** structure. A non-zero value indicates that the Ethernet interface will receive all frames on the network. The default value of this flag is zero.

DLIOCSPROMISC
Allows a privileged process to toggle the current state of the promiscuous flag. When the flag is non-zero, the driver captures all frames from the network. Processes that are bound to the promiscuous SAP, or to an SAP that matches the type field of the received frame, receive a copy of the frame.

DLIOCGETMULTI
Returns the current list of multicast addresses (if it exists).

DLIOCADDMULTI
Allows a privileged process to add a new multicast address and enable its reception. A 6-byte buffer pointing to the multicast address must be passed as the parameter.

DLIOCDELMULTI
Allows a privileged process to delete a multicast address by passing a 6-byte multicast address as the parameter.

Configuration

The **wd** driver has four configurable parameters in the **/etc/conf/pack.d/wd/space.c** file. If you change this file, you must rebuild the kernel and reboot the system for the changes to take effect.

The configurable parameters are:

N_SAPS
: Defines the number of SAPs that can be bound at any one time. This value should be only slightly larger than anticipated SAP usage. A typical TCP/IP system requires two SAPs (0x800 and 0x806). If you assign too large a value to this parameter, system performance and memory usage may suffer.

STREAMS_LOG
: Defines whether the driver should log debugging messages to the STREAMS logger for the **strace**(1M) utility to display. The module ID used with **strace** is 2101. A value of 0 indicates that no STREAMS debug messages should be generated. A value of 1 enables STREAMS debug messages to be generated. You can also direct the driver to log messages temporarily by using the kernel debugger to change the value of **wdstrlog** (a 4-byte integer) to 1.

  Use STREAMS tracing only when debugging a network problem, because system performance suffers when full **wd** STREAMS logging is in progress.

IFNAME
: This parameter is important only in a TCP/IP networking environment. It defines the string used in displaying network statistics. This string should match the logical interface name assigned in the **/etc/confnet.d/inet/interfaces** file and with **ifconfig**(1M) commands used in the **/etc/inet/rc.inet** configuration script.

Error Codes

The **wd** driver can return the following error codes:

ENXIO Invalid major number or board is not installed.

ECHRNG
: No minor devices left if configured as a cloned device. Increase **N_SAP** value in **/etc/conf/pack.d/wd/space.c** Invalid minor device number if configured as a non-cloned device.

EPERM An **ioctl** was made without the appropriate privilege.

EINVAL
: An **ioctl** was made that did not supply a required input and/or output buffer.

DL_NOTSUPPORTED
: Requested service primitive is not supported.

DL_BADPRIM
: Unknown service primitive was requested.

DL_OUTSTATE
: **DL_BIND_REQ** was issued when the Stream was bound, or **DL_UNBIND_REQ** or **DL_UNITDATA_REQ** were issued when the Stream was unbound.

`DL_ACCESS`
: An attempt was made to bind to **`PROMISCUOUS_SAP`** with insufficient privilege.

`DL_BOUND`
: The requested SAP is already bound. A privileged process may bind to an already bound SAP.

`DL_NOTINIT`
: **`DL_UNITDATA_REQ`** was issued on an Ethernet board that has gone offline due to an error.

`DL_BADDATA`
: **`DL_UNITDATA_REQ`** was issued with a data size that was either larger than the SPDU maximum or smaller than the SPDU minimum.

Files

`/dev/wd_*`
`/etc/conf/pack.d/wd/space.c`

REFERENCES

`getmsg`(2), **`ifconfig`**(1M), **`ioctl`**(2), **`open`**(2), **`putmsg`**(2), **`strace`**(1M)

NAME

wd7000 – WD7000 FASST2 host adapter subsystem

DESCRIPTION

The WD7000 FASST2 SCSI host adapter subsystem enables SCSI target drivers (such as **sd01**, **st01**, and so on) to communicate on the SCSI bus with target controllers and logical units. This driver implements the Portable Device Interface (PDI) for such PDI target drivers.

It is also possible to access this subsystem directly using the pass-through driver interface. This allows you to issue **sb** control blocks directly to the target controller. To find the appropriate device to use, while any device is being accessed through the target driver for example **sd01**), use the **B_GETDEV ioctl** to get the major and minor numbers of the pass-though node. This node may be created and opened for pass-through use (**SDI_SEND ioctl**).

ioctl Calls

The following **ioctl**(2) commands are supported by this driver:

SDI_SEND
: Sends a pass-through command (SCSI control block) to a target controller, bypassing the associated target driver.

SDI_BRESET
: Resets the SCSI bus.

B_REDT
: Reads the extended Equipped Device Table (EDT) data structure that is stored in the SCSI driver's internal data structure.

B_HA_CNT
: Gets the value of the number of host adapters for which the SCSI driver is configured.

HA_VER
: Determines the Driver Interface Version supported by the driver. It returns the structure *ver_no*, defined in **/usr/include/sys/sdi.h**.

B_GETTYPE
: Returns the bus name (for example, **scsi**) and device driver name of a specific device.

Files

/usr/include/sys/wd7000.h
/usr/include/sys/scsi.h
/usr/include/sys/sdi.h
/usr/include/sys/sdi_edt.h
/etc/conf/pack.d/wd7000/space.c/fP

REFERENCES

adsc(7), **dpt**(7), **ioctl**(2), **mcis**(7), **sc01**(7), **sd01**(7), **st01**(7), **sw01**(7)

zero (7)

NAME

zero – source of zeroes

DESCRIPTION

A zero special file is a source of zeroed unnamed memory.

Reads from a zero special file always return a buffer full of zeroes. The file is of infinite length.

Writes to a zero special file are always successful, but the data written is ignored.

Mapping a zero special file creates a zero-initialized unnamed memory object of a length equal to the length of the mapping and rounded up to the nearest page size as returned by **sysconf**. Multiple processes can share such a zero special file object provided a common ancestor mapped the object **MAP_SHARED**.

Files

/dev/zero

REFERENCES

fork(2), **mmap**(2), **sysconf**(3C)

Reference Manual Index

The Permuted Index that follows is a list of keywords, alphabetized in the second of three columns, together with the context in which each keyword is found. The manual page that produced an entry is listed in the right column.

Entries are identified with their section numbers shown in parentheses. This is important because there is considerable duplication of names among the sections, arising principally from commands and functions that exist only to exercise a particular system call.

The index is produced by rotating the NAME section of each manual page to alphabetize each keyword in it. Words that cannot fit in the middle column are rotated into the left column. If the entry is still too long, some words are omitted, and their omission is indicated with a slash ("/").

How the Permuted Index Is Created

Many users find that understanding a few things about how the permuted index is created helps them to read it more effectively and clarifies what kind of information can and cannot be obtained from it.

The basic building block for the index is the one-line description given in the NAME line on the top of each manual page. For example, this is what the top of the **mountall**(1M) manual page looks like:

---

mountall(1M) **mountall(1M)**

NAME

`mountall, umountall` – mount, unmount multiple file systems

Each NAME line includes:

- the command, file format, system call or other utility for which the manual page is named (this is the primary utility; **mountall** is the primary utility in the example)
- secondary utilities, which are also described on that manual page and do not have a separate manual page of their own (**umountall** is a secondary utility in the example)

- a brief description of the utility function(s)

For each manual page NAME line, the indexing software generates several index entries, generally one entry for each keyword in the phrase. The middle column of the index is alphabetized on these keywords.

For:

NAME

`mountall, umountall` – mount, unmount multiple file systems

This is generated:

```
       mount, unmount multiple   file systems. /umountall: ..................... mountall(1M)
systems. mountall, umountall:    mount, unmount multiple file .................. mountall(1M)
 unmount multiple file systems.  mountall, umountall: mount, ................... mountall(1M)
  /umountall: mount, unmount     multiple file systems. ........................ mountall(1M)
  mount, unmount multiple file   systems. mountall, umountall: ................. mountall(1M)
        multiple file/ mountall, umountall: mount, unmount ..................... mountall(1M)
 mountall, umountall: mount,     unmount multiple file systems. ................ mountall(1M)
```

How to Use the Index

Look in the middle column of the index for the word of interest. Then read the complete phrase by starting with the utility name, which may appear in the left or middle column. Utility names are followed by a colon.

The NAME line phrase is contained in the two columns, with long phrases wrapping around to the beginning of the left column. The right column of the index provides the manual page name and section number.

A slash (/) sometimes appears in the index entry to indicate that space limitations were exceeded and one or more words from the phrase were deleted.

Permuted Index

adsc Adaptec	1542A SCSI host adapter subsystem	adsc(7)
ee16 EtherExpress	16 Ethernet Adapter Driver	ee(7)
el16 EtherLink	16 Ethernet Adapter Driver	el(7)
ie6	3C503 3Com Ethernet Driver	ie6(7)
ie6 3C503	3Com Ethernet Driver	ie6(7)
compatibility module ttcompat V7,	4BSD and XENIX STREAMS	ttcompat(7)
acct per-process	accounting file format	acct(4)
holidays	accounting file	holidays(4)
format	acct per-process accounting file	acct(4)
subsystem adsc	Adaptec 1542A SCSI host adapter	adsc(7)
ee16 EtherExpress 16 Ethernet	Adapter Driver	ee(7)
el16 EtherLink 16 Ethernet	Adapter Driver	el(7)
mcis MCIS SCSI host	adapter driver	mcis(7)
adsc Adaptec 1542A SCSI host	adapter subsystem	adsc(7)
DCD Direct-Coupled Disk host	adapter Subsystem	DCD(7)
dpt DPT PM2012 SCSI host	adapter subsystem	dpt(7)
wd7000 WD7000 FASST2 host	adapter subsystem	wd7000(7)
ARP	Address Resolution Protocol	ARP(7)
domain ethers Ethernet	address to hostname database or	ethers(4)
aliases, addresses, forward (BSD)	addresses and aliases for sendmail	aliases(4)
and aliases for sendmail aliases,	addresses, forward (BSD) addresses	aliases(4)
	admin installation defaults file	admin(4)
sad STREAMS	Administrative Driver	sad(7)
adapter subsystem	adsc Adaptec 1542A SCSI host	adsc(7)
and match/ regexp: compile, step,	advance regular expression compile	regexp(5)
alp	algorithm pool management module	alp(7)
pathalias	alias file for FACE	pathalias(4)
addresses and aliases for sendmail	aliases, addresses, forward (BSD)	aliases(4)
forward (BSD) addresses and	aliases for sendmail /addresses,	aliases(4)
services Internet services and	aliases	services(4)
module	alp algorithm pool management	alp(7)
Format) files	a.out ELF (Executable and Linking	a.out(4)
	ar archive file format	ar(4)
.ott FACE object	architecture information	.ott(4)
ar	archive file format	ar(4)
	archives device header file	archives(4)
stdarg handle variable	argument list	stdarg(5)
varargs handle variable	argument list	varargs(5)
	ARP Address Resolution Protocol	ARP(7)
ascii map of	ASCII character set	ascii(5)
	ascii map of ASCII character set	ascii(5)
	asyc asynchronous serial port	asyc(7)
asyc	asynchronous serial port	asyc(7)
loginlog log of failed login	attempts	loginlog(4)
OlIcValues Input Context	attribute names and value pairs	OlIcValues(4)
OlImValues a list of IM	attributes	OlImValues(4)
bootparams boot parameter data	base	bootparams(4)

hosts host name data	base	hosts(4)
netmasks network mask data	base	netmasks(4)
networks network name data	base	networks(4)
protocols protocol name data	base	protocols(4)
rpc rpc program number data	base	rpc(4)
signal	base signals	signal(5)
/reserved major numbers for	base system device drivers	res_major(4)
terminfo terminal capability data	base N)	terminfo(4
kmem perform I/O on kernel memory	based on symbol name	kmem(7)
fs (bfs) format of the	bfs file system volume	fs(4)
inode	(bfs) format of a bfs i-node	inode(4)
volume fs	(bfs) format of the bfs file system	fs(4)
inode (bfs) format of a	bfs i-node	inode(4)
for the ckbinarsys command	binarsys remote system information	binarsys(4)
and defaults file (scalable and	bitmapped) /XWIN font configuration	Xwinfont(4)
netdrivers data file for networking	boards to protocols mappings	netdrivers(4)
	boot boot options	boot(4)
dump	boot dump timeout file	dump(4)
boot	boot options	boot(4)
bootparams	boot parameter data base	bootparams(4)
	bootparams boot parameter data base	bootparams(4)
aliases, addresses, forward	(BSD) addresses and aliases for/	aliases(4)
syslogd system log/ syslog.conf	(BSD) configuration file for	syslog.conf(4)
me	(BSD) macros for formatting papers	me(5)
for eqn eqnchar	(BSD) special character definitions	eqnchar(5)
ms	(BSD) text formatting macros	ms(5)
mouse mouse device driver for	bus, serial, and PS/2 mouse devices	mouse(7)
data returned by stat system	call stat (XENIX)	stat(4)
stat data returned by stat system	call	stat(5)
N) terminfo terminal	capability data base	terminfo(4
pnch file format for	card images	pnch(4)
/(cdfs) format of CD-ROM file system	(cdfs) directory data structure	dir(4)
fs (cdfs) format of a	cdfs file system	fs(4)
fs	(cdfs) format of a cdfs file system	fs(4)
inode	(cdfs) format of a cdfs inode	inode(4)
(cdfs) directory data/ dir	(cdfs) format of CD-ROM file system	dir(4)
inode (cdfs) format of a	cdfs inode	inode(4)
data/ dir (cdfs) format of	CD-ROM file system (cdfs) directory	dir(4)
sc01	CD-ROM Target Driver	sc0(7)
eqnchar (BSD) special	character definitions for eqn	eqnchar(5)
ascii map of ASCII	character set	ascii(5)
pkginfo package	characteristics file	pkginfo(4)
setinfo set	characteristics file	setinfo(4)
remote system information for the	ckbinarsys command binarsys	binarsys(4)
rtc real time	clock interface	rtc(7)
pair on a STREAMS driver	clone open any major/minor device	clone(7)
cram	CMOS RAM interface	cram(7)
file header for common object file	(COFF) filehdr	filehdr(4)
Xwincmaps XWIN	color map file	Xwincmaps(4)
information for the ckbinarsys	command binarsys remote system	binarsys(4)
form description file for menu(1)	command menu	menu(4)
of mail mailsurr surrogate	commands for routing and transport	mailsurr(4)

dfstab file containing	commands for sharing resources	dfstab(4)
streamio STREAMS ioctl	commands	streamio(7)
filehdr file header for	common object file (COFF)	filehdr(4)
ttcompat V7, 4BSD and XENIX STREAMS	compatibility module	ttcompat(7)
compver	compatible versions file	compver(4)
/step, advance regular expression	compile and match routines	regexp(5)
expression compile and/ regexp:	compile, step, advance regular	regexp(5)
term format of	compiled term file	term(4)
	compver compatible versions file	compver(4)
Sassign	configurable device variables	Sassign(4)
(scalable and/ Xwinfont XWIN font	configuration and defaults file	Xwinfont(4)
netconfig network	configuration database	netconfig(4)
mass-storage and SCSI/ disk.cfg	configuration defaults for	disk.cfg(4)
routines resolv.conf	configuration file for name server	resolv.conf(4)
Information Service (NIS)/ updaters	configuration file for Network	updaters(4)
TCP/IP strcf STREAMS	Configuration File for STREAMS	strcf(4)
system log/ syslog.conf (BSD)	configuration file for syslogd	syslog.conf(4)
Xwinconfig XWIN	configuration file	Xwinconfig(4)
mass-storage devices tc.index	configuration index file for	tc.index(4)
kernel module Master generic	configuration information for a	Master(4)
kernel/ System system-specific	configuration information for a	System(4)
Internet network interface	configuration parameters interface	interface(4)
module data structure/ Space.c	configuration-dependent kernel	Space.c(4)
line discipline for unique stream	connections connld	connld(7)
stream connections	connld line discipline for unique	connld(7)
display system	console display	display(7)
console STREAMS-based	console interface	console(7)
keyboard system	console keyboard	keyboard(7)
interface	console STREAMS-based console	console(7)
langinfo language information	constants	langinfo(5)
file for implementation-specific	constants limits header	limits(4)
math math functions and	constants	math(5)
unistd header file for symbolic	constants	unistd(4)
resources dfstab file	containing commands for sharing	dfstab(4)
information for/ ttydefs file	contains terminal line settings	ttydefs(4)
pkgmap package	contents description file	pkgmap(4)
pairs OlIcValues Input	Context attribute names and value	OlIcValues(4)
ucontext user	context	ucontext(5)
vxfsio vxfs file system	control functions	vxfsio(7)
ICMP Internet	Control Message Protocol	ICMP(7)
fcntl file	control options	fcntl(5)
TCP Internet Transmission	Control Protocol	TCP(7)
tty	controlling terminal interface	tty(7)
term	conventional names for terminals	term(5)
iconv code set	conversion tables	iconv(5)
timod Transport Interface	cooperating STREAMS module	timod(7)
	core core image file	core(4)
core	core image file	core(4)
mem, kmem	core memory	mem(7)
	cram CMOS RAM interface	cram(7)
crontab and at	cron, queuedefs option files for	cron(4)
cron, queuedefs option files for	crontab and at	cron(4)

Device Interface (PDI) WORM Target	Driver sw01 Portable	sw0(7)
sxt pseudo-device	driver	sxt(7)
wd Western Digital WD8003 Ethernet	Driver	wd(7)
interface to EUC handling tty	drivers and modules /generic	eucioctl(5)
numbers for base system device	drivers res_major reserved major	res_major(4)
	dump boot dump timeout file	dump(4)
dump boot	dump timeout file	dump(4)
Adapter Driver	ee16 EtherExpress 16 Ethernet	ee(7)
Driver	el16 EtherLink 16 Ethernet Adapter	el(7)
files a.out	ELF (Executable and Linking Format)	a.out(4)
ptem STREAMS pseudo-terminal	emulation module	ptem(7)
uuencode format of an	encoded uuencode file	uuencode(4)
Init inittab	entries for a kernel module	Init(4)
file system independent directory	entry dirent	dirent(4)
utmp, wtmp utmp and wtmp	entry formats	utmp(4)
utmpx, wtmpx utmpx and wtmpx	entry formats	utmpx(4)
user-preference variable files for/	.environ, .pref, .variables	.environ(4)
	environ user environment	environ(5)
profile setting up an	environment at login time	profile(4)
environ user	environment	environ(5)
special character definitions for	eqn eqnchar (BSD)	eqnchar(5)
definitions for eqn	eqnchar (BSD) special character	eqnchar(5)
log interface to STREAMS	error logging and event tracing	log(7)
Driver ee16	EtherExpress 16 Ethernet Adapter	ee(7)
Driver el16	EtherLink 16 Ethernet Adapter	el(7)
ee16 EtherExpress 16	Ethernet Adapter Driver	ee(7)
el16 EtherLink 16	Ethernet Adapter Driver	el(7)
database or domain ethers	Ethernet address to hostname	ethers(4)
i596 i596	Ethernet Driver	i596(7)
ie6 3C503 3Com	Ethernet Driver	ie6(7)
imx586 IMXLAN586 Intel	Ethernet Driver	imx586(7)
wd Western Digital WD8003	Ethernet Driver	wd(7)
database or domain	ethers Ethernet address to hostname	ethers(4)
eucioctl generic interface to	EUC handling tty drivers and/	eucioctl(5)
handling tty drivers and modules	eucioctl generic interface to EUC	eucioctl(5)
to STREAMS error logging and	event tracing log interface	log(7)
files a.out ELF	(Executable and Linking Format)	a.out(4)
/compile, step, advance regular	expression compile and match/	regexp(5)
termiox	extended general terminal interface	termiox(7)
user-preference variable files for	FACE /.pref, .variables	.environ(4)
information .ott	FACE object architecture	.ott(4)
pathalias alias file for	FACE	pathalias(4)
loginlog log of	failed login attempts	loginlog(4)
inet Internet protocol	family	inet(7)
wd7000 WD7000	FASST2 host adapter subsystem	wd7000(7)
	fcntl file control options	fcntl(5)
	fd diskette (floppy disk)	fd(7)
	fd file descriptor files	fd(4)
admin installation defaults	file	admin(4)
archives device header	file	archives(4)
file header for common object	file (COFF) filehdr	filehdr(4)
compver compatible versions	file	compver(4)

proc process	file system	proc(4)
lid_and_priv distributed	file system security database	lid_and_priv(4)
mnttab mounted	file system table	mnttab(4)
sharetab shared	file system table	sharetab(4)
fs (bfs) format of the bfs	file system volume	fs(4)
fs (s5) format of s5	file system volume	fs(4)
fs (sfs) format of sfs	file system volume	fs(4)
fs (ufs) format of ufs	file system volume	fs(4)
fs (vxfs) format of vxfs	file system volume	fs(4)
term format of compiled term	file	term(4)
file system packages fstypes	file that registers distributed	fstypes(4)
format of an encoded uuencode	file uuencode	uuencode(4)
Xwincmaps XWIN color map	file	Xwincmaps(4)
Xwinconfig XWIN configuration	file	Xwinconfig(4)
object file (COFF)	filehdr file header for common	filehdr(4)
ELF (Executable and Linking Format)	files a.out	a.out(4)
depend software dependencies	files	depend(4)
fd file descriptor	files	fd(4)
cron, queuedefs option	files for crontab and at	cron(4)
user-preference variable	files for FACE /.variables	.environ(4)
font font description	files for troff and dpost	font(5)
fspec format specification in text	files	fspec(4)
intro introduction to special	files	intro(7)
format of tty device mapping	files mapchan	mapchan(4)
	filesystem file system organization	filesystem(7)
fd diskette	(floppy disk)	fd(7)
file (scalable and/ Xwinfont XWIN	font configuration and defaults	Xwinfont(4)
and dpost font	font description files for troff	font(5)
troff and dpost	font font description files for	font(5)
command menu	form description file for menu(1)	menu(4)
acct per-process accounting file	format	acct(4)
ar archive file	format	ar(4)
a.out ELF (Executable and Linking	Format) files	a.out(4)
pnch file	format for card images	pnch(4)
mkdev file	format for the pdimkdev utility	mkdev(4)
help Desktop help file	format	help(4)
inode (bfs)	format of a bfs i-node	inode(4)
fs (cdfs)	format of a cdfs file system	fs(4)
inode (cdfs)	format of a cdfs inode	inode(4)
inode (sfs)	format of a sfs inode	inode(4)
inode (ufs)	format of a ufs inode	inode(4)
inode (vxfs)	format of a vxfs inode	inode(4)
uuencode	format of an encoded uuencode file	uuencode(4)
inode (s5)	format of an s5 i-node	inode(4)
directory data/ dir (cdfs)	format of CD-ROM file system (cdfs)	dir(4)
term	format of compiled term file	term(4)
dir (s5)	format of s5 directories	dir(4)
fs (s5)	format of s5 file system volume	fs(4)
sccsfile	format of SCCS file	sccsfile(4)
fs (sfs)	format of sfs file system volume	fs(4)
volume fs (bfs)	format of the bfs file system	fs(4)
mapchan	format of tty device mapping files	mapchan(4)

by system and by user	hosts.equiv, .rhosts trusted hosts	hosts.equiv(4)
i596	i596 Ethernet Driver	i596(7)
	i596 i596 Ethernet Driver	i596(7)
ibmtok	IBM Token Ring Driver	ibmtok(7)
	ibmtok IBM Token Ring Driver	ibmtok(7)
Protocol	ICMP Internet Control Message	ICMP(7)
	iconv code set conversion tables	iconv(5)
issue issue	identification file	issue(4)
	ie6 3C503 3Com Ethernet Driver	ie6(7)
OlImValues a list of	IM attributes	OlImValues(4)
core core	image file	core(4)
pnch file format for card	images	pnch(4)
limits header file for	implementation-specific constants	limits(4)
Driver	imx586 IMXLAN586 Intel Ethernet	imx586(7)
imx586	IMXLAN586 Intel Ethernet Driver	imx586(7)
mechanism definitions privilege	include file for privilege	privilege(5)
privilege definitions priv	include file for user–level	priv(5)
dirent file system	independent directory entry	dirent(4)
tc.index configuration	index file for mass-storage devices	tc.index(4)
	inet Internet protocol family	inet(7)
database	inetd.conf Internet servers	inetd.conf(4)
langinfo language	information constants	langinfo(5)
copyright copyright	information file	copyright(4)
prototype package	information file	prototype(4)
Master generic configuration	information for a kernel module	Master(4)
/system-specific configuration	information for a kernel module	System(4)
mailcnfg initialization	information for mail and rmail	mailcnfg(4)
command binarsys remote system	information for the ckbinarsys	binarsys(4)
contains terminal line settings	information for ttymon /file	ttydefs(4)
.ott FACE object architecture	information	.ott(4)
and directory/ ypfiles the Network	Information Service (NIS) database	ypfiles(4)
/configuration file for Network	Information Service (NIS) updating	updaters(4)
siginfo signal generation	information	siginfo(5)
module	Init inittab entries for a kernel	Init(4)
inittab script for	init	inittab(4)
and rmail mailcnfg	initialization information for mail	mailcnfg(4)
/kernel module data structure	initializations	Space.c(4)
Init	inittab entries for a kernel module	Init(4)
	inittab script for init	inittab(4)
	inode (bfs) format of a bfs i-node	inode(4)
	inode (cdfs) format of a cdfs inode	inode(4)
inode (bfs) format of a bfs	i-node	inode(4)
inode (cdfs) format of a cdfs	inode	inode(4)
inode (s5) format of an s5	i-node	inode(4)
inode (sfs) format of a sfs	inode	inode(4)
inode (ufs) format of a ufs	inode	inode(4)
inode (vxfs) format of a vxfs	inode	inode(4)
	inode (s5) format of an s5 i-node	inode(4)
	inode (sfs) format of a sfs inode	inode(4)
	inode (ufs) format of a ufs inode	inode(4)
	inode (vxfs) format of a vxfs inode	inode(4)
value pairs OlIcValues	Input Context attribute names and	OlIcValues(4)

admin	installation defaults file	admin(4)
imx586 IMXLAN586	Intel Ethernet Driver	imx586(7)
interface Internet network	interface configuration parameters	interface(4)
console STREAMS-based console	interface	console(7)
module timod Transport	Interface cooperating STREAMS	timod(7)
cram CMOS RAM	interface	cram(7)
interface configuration parameters	interface Internet network	interface(4)
lo software loopback network	interface	lo(7)
lp parallel port	interface	lp(7)
st01 Portable Device	Interface (PDI) tape target driver	st0(7)
sw01 Portable Device	Interface (PDI) WORM Target Driver	sw0(7)
STREAMS module tirdwr Transport	Interface read/write interface	tirdwr(7)
rtc real time clock	interface	rtc(7)
Transport Interface read/write	interface STREAMS module tirdwr	tirdwr(7)
termio general terminal	interface	termio(7)
termiox extended general terminal	interface	termiox(7)
drivers and/ eucioctl generic	interface to EUC handling tty	eucioctl(5)
and event tracing log	interface to STREAMS error logging	log(7)
tty controlling terminal	interface	tty(7)
of Internet Protocol network	interfaces if general properties	if(7)
ICMP	Internet Control Message Protocol	ICMP(7)
configuration parameters interface	Internet network interface	interface(4)
inet	Internet protocol family	inet(7)
IP	Internet Protocol	IP(7)
if general properties of	Internet Protocol network/	if(7)
inetd.conf	Internet servers database	inetd.conf(4)
services	Internet services and aliases	services(4)
Protocol TCP	Internet Transmission Control	TCP(7)
UDP	Internet User Datagram Protocol	UDP(7)
	intro introduction to miscellany	intro(5)
	intro introduction to special files	intro(7)
intro	introduction to miscellany	intro(5)
intro	introduction to special files	intro(7)
symbol name kmem perform	I/O on kernel memory based on	kmem(7)
streamio STREAMS	ioctl commands	streamio(7)
sockets sockio	ioctls that operate directly on	sockio(7)
	IP Internet Protocol	IP(7)
issue	issue identification file	issue(4)
	issue issue identification file	issue(4)
module	kbd generalized string translation	kbd(7)
kmem perform I/O on	kernel memory based on symbol name	kmem(7)
Space.c configuration-dependent	kernel module data structure/	Space.c(4)
Init inittab entries for a	kernel module	Init(4)
configuration information for a	kernel module Master generic	Master(4)
Stubs.c stubs for	kernel module symbols	Stubs.c(4)
script Sd	kernel module system shutdown	Sd(4)
configuration information for a	kernel module /system-specific	System(4)
publickey public	key database	publickey(4)
keyboard system console	keyboard	keyboard(7)
	keyboard system console keyboard	keyboard(7)
mem,	kmem core memory	mem(7)
based on symbol name	kmem perform I/O on kernel memory	kmem(7)

constants	langinfo language information	langinfo(5)
nl_types native	language data types	nl_types(5)
langinfo	language information constants	langinfo(5)
strftime	language-specific strings	strftime(4)
line discipline module	ldterm standard STREAMS terminal	ldterm(7)
system security database	lid_and_priv distributed file	lid_and_priv(4)
implementation-specific constants	limits header file for	limits(4)
connections connld	line discipline for unique stream	connld(7)
ldterm standard STREAMS terminal	line discipline module	ldterm(7)
ttydefs file contains terminal	line settings information for/	ttydefs(4)
a.out ELF (Executable and	Linking Format) files	a.out(4)
ttysrch directory search	list for ttymap and ttyname	ttysrch(4)
OlImValues a	list of IM attributes	OlImValues(4)
stdarg handle variable argument	list	stdarg(5)
varargs handle variable argument	list	varargs(5)
interface	lo software loopback network	lo(7)
parameters stune	local system settings for tunable	stune(4)
file for syslogd system	log daemon /(BSD) configuration	syslog.conf(4)
logging and event tracing	log interface to STREAMS error	log(7)
loginlog	log of failed login attempts	loginlog(4)
log interface to STREAMS error	logging and event tracing	log(7)
loginlog log of failed	login attempts	loginlog(4)
netrc file for ftp remote	login data	netrc(4)
login	login default file	login(4)
	login login default file	login(4)
setting up an environment at	login time profile	profile(4)
attempts	loginlog log of failed login	loginlog(4)
lo software	loopback network interface	lo(7)
ticlts, ticots, ticotsord	loopback transport providers	ticlts(7)
	lp parallel port interface	lp(7)
values	machine-dependent values	values(5)
me (BSD)	macros for formatting papers	me(5)
ms (BSD) text formatting	macros	ms(5)
pages man	macros to format Reference Manual	man(5)
initialization information for	mail and rmail mailcnfg	mailcnfg(4)
for routing and transport of	mail mailsurr surrogate commands	mailsurr(4)
for mail and rmail	mailcnfg initialization information	mailcnfg(4)
routing and transport of mail	mailsurr surrogate commands for	mailsurr(4)
device drivers res_major reserved	major numbers for base system	res_major(4)
STREAMS driver clone open any	major/minor device pair on a	clone(7)
alp algorithm pool	management module	alp(7)
man macros to format Reference	Manual pages	man(5)
Xwincmaps XWIN color	map file	Xwincmaps(4)
ascii	map of ASCII character set	ascii(5)
mapping files	mapchan format of tty device	mapchan(4)
mapchan format of tty device	mapping files	mapchan(4)
for networking boards to protocols	mappings netdrivers data file	netdrivers(4)
netmasks network	mask data base	netmasks(4)
disk.cfg configuration defaults for	mass-storage and SCSI devices	disk.cfg(4)
configuration index file for	mass-storage devices tc.index	tc.index(4)
Remote File Sharing name server	master file rfmaster	rfmaster(4)
information for a kernel module	Master generic configuration	Master(4)

regular expression compile and	match routines /step, advance	regexp(5)
math	math functions and constants	math(5)
	math math functions and constants	math(5)
	mcis MCIS SCSI host adapter driver	mcis(7)
mcis	MCIS SCSI host adapter driver	mcis(7)
include file for privilege	mechanism definitions privilege	privilege(5)
	mem, kmem core memory	mem(7)
kmem perform I/O on kernel	memory based on symbol name	kmem(7)
mem, kmem core	memory	mem(7)
menu(1) command	menu form description file for	menu(4)
menu form description file for	menu(1) command	menu(4)
ICMP Internet Control	Message Protocol	ICMP(7)
intro introduction to	miscellany	intro(5)
utility	mkdev file format for the pdimkdev	mkdev(4)
	mnttab mounted file system table	mnttab(4)
pckt STREAMS Packet	Mode module	pckt(7)
alp algorithm pool management	module	alp(7)
configuration-dependent kernel	module data structure/ Space.c	Space.c(4)
Init inittab entries for a kernel	module	Init(4)
kbd generalized string translation	module	kbd(7)
STREAMS terminal line discipline	module ldterm standard	ldterm(7)
information for a kernel	module /generic configuration	Master(4)
pckt STREAMS Packet Mode	module	pckt(7)
STREAMS pseudo-terminal emulation	module ptem	ptem(7)
Stubs.c stubs for kernel	module symbols	Stubs.c(4)
Sd kernel	module system shutdown script	Sd(4)
information for a kernel	module /configuration	System(4)
Interface cooperating STREAMS	module timod Transport	timod(7)
read/write interface STREAMS	module tirdwr Transport Interface	tirdwr(7)
and XENIX STREAMS compatibility	module ttcompat V7, 4BSD	ttcompat(7)
to EUC handling tty drivers and	modules eucioctl generic interface	eucioctl(5)
mnttab	mounted file system table	mnttab(4)
serial, and PS/2 mouse/ mouse	mouse device driver for bus,	mouse(7)
driver for bus, serial, and PS/2	mouse devices mouse mouse device	mouse(7)
serial, and PS/2 mouse devices	mouse mouse device driver for bus,	mouse(7)
	ms (BSD) text formatting macros	ms(5)
	Mtune tunable parameter definitions	Mtune(4)
data base	N) terminfo terminal capability	terminfo(4
hosts host	name data base	hosts(4)
networks network	name data base	networks(4)
protocols protocol	name data base	protocols(4)
on kernel memory based on symbol	name kmem perform I/O	kmem(7)
rfmaster Remote File Sharing	name server master file	rfmaster(4)
resolv.conf configuration file for	name server routines	resolv.conf(4)
OlIcValues Input Context attribute	names and value pairs	OlIcValues(4)
term conventional	names for terminals	term(5)
nl_types	native language data types	nl_types(5)
database	netconfig network configuration	netconfig(4)
boards to protocols mappings	netdrivers data file for networking	netdrivers(4)
	netmasks network mask data base	netmasks(4)
data	netrc file for ftp remote login	netrc(4)
netconfig	network configuration database	netconfig(4)

emulation module	ptem STREAMS pseudo-terminal	ptem(7)
	pty STREAMS pseudo-terminal driver	pty(7)
publickey	public key database	publickey(4)
	publickey public key database	publickey(4)
and at cron,	queuedefs option files for crontab	cron(4)
cram CMOS	RAM interface	cram(7)
	Rc system startup script	Rc(4)
tirdwr Transport Interface	read/write interface STREAMS module	tirdwr(7)
rtc	real time clock interface	rtc(7)
table rt_dptbl	real-time dispatcher parameter	rt_dptbl(4)
man macros to format	Reference Manual pages	man(5)
regular expression compile and/	regexp: compile, step, advance	regexp(5)
packages fstypes file that	registers distributed file system	fstypes(4)
regexp: compile, step, advance	regular expression compile and/	regexp(5)
master file rfmaster	Remote File Sharing name server	rfmaster(4)
netrc file for ftp	remote login data	netrc(4)
ckbinarsys command binarsys	remote system information for the	binarsys(4)
space disk space	requirement file	space(4)
setsize disk space	requirements file	setsize(4)
for base system device drivers	res_major reserved major numbers	res_major(4)
ARP Address	Resolution Protocol	ARP(7)
name server routines	resolv.conf configuration file for	resolv.conf(4)
containing commands for sharing	resources dfstab file	dfstab(4)
stat (XENIX) data	returned by stat system call	stat(4)
stat data	returned by stat system call	stat(5)
server master file	rfmaster Remote File Sharing name	rfmaster(4)
by user hosts.equiv,	.rhosts trusted hosts by system and	hosts.equiv(4)
ibmtok IBM Token	Ring Driver	ibmtok(7)
information for mail and	rmail mailcnfg initialization	mailcnfg(4)
expression compile and match	routines /step, advance regular	regexp(5)
configuration file for name server	routines resolv.conf	resolv.conf(4)
mailsurr surrogate commands for	routing and transport of mail	mailsurr(4)
supporting for packet network	routing routing system	routing(4)
packet network routing	routing system supporting for	routing(4)
rpc	rpc program number data base	rpc(4)
	rpc rpc program number data base	rpc(4)
	rtc real time clock interface	rtc(7)
parameter table	rt_dptbl real-time dispatcher	rt_dptbl(4)
dir (s5) format of	s5 directories	dir(4)
fs (s5) format of	s5 file system volume	fs(4)
inode	(s5) format of an s5 i-node	inode(4)
dir	(s5) format of s5 directories	dir(4)
volume fs	(s5) format of s5 file system	fs(4)
inode (s5) format of an	s5 i-node	inode(4)
	sad STREAMS Administrative Driver	sad(7)
variables	Sassign configurable device	Sassign(4)
	sc01 CD-ROM Target Driver	sc0(7)
configuration and defaults file	(scalable and bitmapped) /XWIN font	Xwinfont(4)
sccsfile format of	SCCS file	sccsfile(4)
	sccsfile format of SCCS file	sccsfile(4)
inittab	script for init	inittab(4)
Rc system startup	script	Rc(4)

eqn eqnchar (BSD)	special character definitions for	eqnchar(5)
intro introduction to	special files	intro(7)
fspec format	specification in text files	fspec(4)
getty gettydefs	speed and terminal settings used by	gettydefs(4)
(PDI) tape target driver	st01 Portable Device Interface	st0(7)
discipline module ldterm	standard STREAMS terminal line	ldterm(7)
Rc system	startup script	Rc(4)
call	stat data returned by stat system	stat(5)
stat (XENIX) data returned by	stat system call	stat(4)
stat data returned by	stat system call	stat(5)
system call	stat (XENIX) data returned by stat	stat(4)
wstat wait	status	wstat(5)
list	stdarg handle variable argument	stdarg(5)
compile and match/ regexp: compile,	step, advance regular expression	regexp(5)
for STREAMS TCP/IP	strcf STREAMS Configuration File	strcf(4)
connld line discipline for unique	stream connections	connld(7)
	streamio STREAMS ioctl commands	streamio(7)
sad	STREAMS Administrative Driver	sad(7)
ttcompat V7, 4BSD and XENIX	STREAMS compatibility module	ttcompat(7)
STREAMS TCP/IP strcf	STREAMS Configuration File for	strcf(4)
any major/minor device pair on a	STREAMS driver clone open	clone(7)
tracing log interface to	STREAMS error logging and event	log(7)
streamio	STREAMS ioctl commands	streamio(7)
Transport Interface cooperating	STREAMS module timod	timod(7)
Interface read/write interface	STREAMS module tirdwr Transport	tirdwr(7)
pckt	STREAMS Packet Mode module	pckt(7)
pty	STREAMS pseudo-terminal driver	pty(7)
module ptem	STREAMS pseudo-terminal emulation	ptem(7)
STREAMS Configuration File for	STREAMS TCP/IP strcf	strcf(4)
module ldterm standard	STREAMS terminal line discipline	ldterm(7)
console	STREAMS-based console interface	console(7)
	strftime language-specific strings	strftime(4)
kbd generalized	string translation module	kbd(7)
strftime language-specific	strings	strftime(4)
file system (cdfs) directory data	structure /(cdfs) format of CD-ROM	dir(4)
/kernel module data	structure initializations	Space.c(4)
(NIS) database and directory	structure /Information Service	ypfiles(4)
Stubs.c	stubs for kernel module symbols	Stubs.c(4)
symbols	Stubs.c stubs for kernel module	Stubs.c(4)
tunable parameters	stune local system settings for	stune(4)
su	su options file	su(4)
	su su options file	su(4)
Adaptec 1542A SCSI host adapter	subsystem adsc	adsc(7)
Direct-Coupled Disk host adapter	Subsystem DCD	DCD(7)
dpt DPT PM2012 SCSI host adapter	subsystem	dpt(7)
wd7000 WD7000 FASST2 host adapter	subsystem	wd7000(7)
routing routing system	supporting for packet network	routing(4)
transport of mail mailsurr	surrogate commands for routing and	mailsurr(4)
(PDI) WORM Target Driver	sw01 Portable Device Interface	sw0(7)
	sxt pseudo-device driver	sxt(7)
I/O on kernel memory based on	symbol name kmem perform	kmem(7)
unistd header file for	symbolic constants	unistd(4)

Stubs.c stubs for kernel module	symbols	Stubs.c(4)
file for syslogd system log daemon	syslog.conf (BSD) configuration	syslog.conf(4)
/(BSD) configuration file for	syslogd system log daemon	syslog.conf(4)
.rhosts trusted hosts by	system and by user hosts.equiv,	hosts.equiv(4)
stat (XENIX) data returned by stat	system call	stat(4)
stat data returned by stat	system call	stat(5)
dir (cdfs) format of CD-ROM file	system (cdfs) directory data/	dir(4)
display	system console display	display(7)
keyboard	system console keyboard	keyboard(7)
vxfsio vxfs file	system control functions	vxfsio(7)
types primitive	system data types	types(5)
vfstab table of file	system defaults	vfstab(4)
reserved major numbers for base	system device drivers res_major	res_major(4)
fs (cdfs) format of a cdfs file	system	fs(4)
dirent file	system independent directory entry	dirent(4)
ckbinarsys command binarsys remote	system information for the	binarsys(4)
configuration file for syslogd	system log daemon /(BSD)	syslog.conf(4)
filesystem file	system organization	filesystem(7)
that registers distributed file	system packages fstypes file	fstypes(4)
proc process file	system	proc(4)
prf operating	system profiler	prf(7)
lid_and_priv distributed file	system security database	lid_and_priv(4)
parameters stune local	system settings for tunable	stune(4)
Sd kernel module	system shutdown script	Sd(4)
Rc	system startup script	Rc(4)
network routing routing	system supporting for packet	routing(4)
configuration information for a/	System system-specific	System(4)
mnttab mounted file	system table	mnttab(4)
sharetab shared file	system table	sharetab(4)
timezone set default	system time zone	timezone(4)
fs (bfs) format of the bfs file	system volume	fs(4)
fs (s5) format of s5 file	system volume	fs(4)
fs (sfs) format of sfs file	system volume	fs(4)
fs (ufs) format of ufs file	system volume	fs(4)
fs (vxfs) format of vxfs file	system volume	fs(4)
information for a kernel/ System	system-specific configuration	System(4)
mnttab mounted file system	table	mnttab(4)
vfstab	table of file system defaults	vfstab(4)
PrivTable privilege	table	PrivTable(4)
real-time dispatcher parameter	table rt_dptbl	rt_dptbl(4)
sharetab shared file system	table	sharetab(4)
time-sharing dispatcher parameter	table ts_dptbl	ts_dptbl(4)
iconv code set conversion	tables	iconv(5)
Portable Device Interface (PDI)	tape target driver st01	st0(7)
sc01 CD-ROM	Target Driver	sc0(7)
sd01 PDI disk	target driver	sd0(7)
Device Interface (PDI) tape	target driver st01 Portable	st0(7)
Device Interface (PDI) WORM	Target Driver sw01 Portable	sw0(7)
for mass-storage devices	tc.index configuration index file	tc.index(4)
Protocol	TCP Internet Transmission Control	TCP(7)
Configuration File for STREAMS	TCP/IP strcf STREAMS	strcf(4)
terminals	term conventional names for	term(5)

term format of compiled	term file	term(4)
	term format of compiled term file	term(4)
N) terminfo	terminal capability data base	terminfo(4
termio general	terminal interface	termio(7)
termiox extended general	terminal interface	termiox(7)
tty controlling	terminal interface	tty(7)
ldterm standard STREAMS	terminal line discipline module	ldterm(7)
for ttymon ttydefs file contains	terminal line settings information	ttydefs(4)
gettydefs speed and	terminal settings used by getty	gettydefs(4)
term conventional names for	terminals	term(5)
base N)	terminfo terminal capability data	terminfo(4
	termio general terminal interface	termio(7)
interface	termiox extended general terminal	termiox(7)
fspec format specification in	text files	fspec(4)
ms (BSD)	text formatting macros	ms(5)
transport providers	ticlts, ticots, ticotsord loopback	ticlts(7)
transport providers ticlts,	ticots, ticotsord loopback	ticlts(7)
providers ticlts, ticots,	ticotsord loopback transport	ticlts(7)
dump boot dump	timeout file	dump(4)
table ts_dptbl	time-sharing dispatcher parameter	ts_dptbl(4)
zone	timezone set default system time	timezone(4)
cooperating STREAMS module	timod Transport Interface	timod(7)
read/write interface STREAMS/	tirdwr Transport Interface	tirdwr(7)
ibmtok IBM	Token Ring Driver	ibmtok(7)
to STREAMS error logging and event	tracing log interface	log(7)
kbd generalized string	translation module	kbd(7)
TCP Internet	Transmission Control Protocol	TCP(7)
STREAMS module timod	Transport Interface cooperating	timod(7)
interface STREAMS module tirdwr	Transport Interface read/write	tirdwr(7)
surrogate commands for routing and	transport of mail mailsurr	mailsurr(4)
ticlts, ticots, ticotsord loopback	transport providers	ticlts(7)
font font description files for	troff and dpost	font(5)
hosts.equiv, .rhosts	trusted hosts by system and by user	hosts.equiv(4)
parameter table	ts_dptbl time-sharing dispatcher	ts_dptbl(4)
compatibility module	ttcompat V7, 4BSD and XENIX STREAMS	ttcompat(7)
	tty controlling terminal interface	tty(7)
mapchan format of	tty device mapping files	mapchan(4)
generic interface to EUC handling	tty drivers and modules eucioctl	eucioctl(5)
settings information for ttymon	ttydefs file contains terminal line	ttydefs(4)
ttysrch directory search list for	ttymap and ttyname	ttysrch(4)
line settings information for	ttymon /file contains terminal	ttydefs(4)
search list for ttymap and	ttyname ttysrch directory	ttysrch(4)
ttymap and ttyname	ttysrch directory search list for	ttysrch(4)
Mtune	tunable parameter definitions	Mtune(4)
stune local system settings for	tunable parameters	stune(4)
nl_types native language data	types	nl_types(5)
	types primitive system data types	types(5)
types primitive system data	types	types(5)
	ucontext user context	ucontext(5)
	UDP Internet User Datagram Protocol	UDP(7)
dir (ufs) format of	ufs directories	dir(4)
fs (ufs) format of	ufs file system volume	fs(4)

R E F E R E N C E

This definitive reference set describes every UNIX® System V Release 4 command, system call, library function, and file format, including the BSD and XENIX® variants unified under Release 4. Written by UNIX System Laboratories, source of the UNIX System V operating system, this set includes the following manuals:

- The two-volume ***Command Reference*** describes all user and administrative commands in the UNIX system, including file handling, basic networking, shell programming, and system management commands.

- The ***Operating System API Reference*** describes UNIX system calls and library functions, including C language, math, networking, and specialized libraries.

- The ***Windowing System API Reference*** describes graphical and character-based libraries, critical elements for building powerful user-interfaces on workstations, X, and character terminals.

- The ***System Files and Devices Reference*** describes the file formats for important system files, such as password, hosts, system initialization, and special (device) files.

- The ***Device Driver Reference,*** consists of two parts. The first part describes the Device Driver Interface/Driver-Kernel Interface (DDI/DKI). The DDI/DKI is a mature interface between drivers and the rest of the kernel. The second part describes routines of the Portable Device Interface (PDI). The PDI is a newer interface for block-oriented devices that emphasizes the separation of hardware-dependent and hardware-independent pieces of drivers.

UNIX
PRESS

A Prentice Hall Title

ISBN 0-13-017682-6

UNIX
Documentation