CTIX™ OPERATING SYSTEM MANUAL

Version C
Volume 4

This manual was prepared on a Convergent Technologies S/320 Computer System and was printed on an Apple LaserWriter II Laser Printer.

Second Edition (November 1989) 09-02265-01

TABLE OF CONTENTS: VOLUME 4

5. Miscellaneous Facilities

6. Games

7. Special Files

HOW TO USE THIS MANUAL

This second edition of the *CTIX Operating System Manual, Version C,* describes the commands, system calls, libraries, data files, and device interfaces that make up the CTIX Operating System for S/Series Computer Systems. This manual should always be your starting point when you need to find the documentation for a CTIX feature with which you are unfamiliar.

The manual consists of a large number of short entries, sometimes called "the *man* pages," after the command that accesses the entries when they are kept online. Each entry briefly documents some feature of CTIX. Some features require longer documentation than an entry in this manual; such features have an entry that outlines the feature and cross-references the manual that documents the feature fully. Entries that do not refer to other manuals are self-contained and are the final word on the features they describe.

Organization of the manual. The entries are organized into seven sections in four volumes:

Volumes 1 and 2:

1. Commands and Application Programs.

Volume 3:

2. System Calls.
3. Subroutines and Libraries.

Volume 4:

4. File Formats.
5. Miscellaneous Facilities.
6. Games.
7. Special Files.

Within each section, entries are alphabetical by title, except for an *intro* entry at the beginning of each section.

Entry Title Conventions. An entry title looks like this example:

```
erf(3M)
|   ||
|   |Entry Type
|   |
|   Section Number
|
Name
```

Name is the name of the entry. *Section Number* indicates the section that contains the entry. In this case, the entry is in Section 3, which is in Volume 2. *Entry Type* appears only on entries that belong to special categories; refer to the section's *intro* entry for an explanation. In this case, a reference to *intro*(3) would tell you that *erf*(3M) describes functions from the Math Library, which the C compiler does not load by default.

Finding the entry you need. To find out which entry you need, refer to the following guides:

- The Permuted Index. This indexes each significant word in each entry's description. It is useful when you have only a general notion what you're looking for. It is also useful when you know the name of the command or function you are interested in, but there is no entry by that name.
- The Table of Contents. This is a simple list of entries, by section, together with the entry descriptions. Volumes 1 and 2 have Tables of Contents for Section 1. Volume 3 has a Table of Contents for Sections 2 and 3. Volume 4 has a Table of Contents for Sections 4 through 7.
- The Table of Related Entries. For Volume 1 only. A table of entries organized so that related entries are grouped together.

Section organization. Each section begins with an *intro* entry, which provides important general information for that section.

Section 1, Commands and Application Programs, describes programs intended to be invoked directly by the user or by command language procedures, as opposed to subroutines, which are intended to be called by the user's programs. Commands generally reside in the directory **/bin** (for **bin**ary programs). Some programs also reside in **/usr/bin**, to save space in **/bin**. These directories are searched automatically by the command interpreter called the *shell*. Commands that were not transported from UNIX System V reside in **/usr/local/bin**; this directory is recommended for locally implemented programs. Some administrative commands reside in **/etc** and various other places. The **/etc** directory is searched automatically if you are logged in as root; otherwise use the full path name given under **SYNOPSIS** or change the **PATH** environment variable to include the command's directory.

Section 2, System Calls, describes the entries into the CTIX kernel, including the C language interfaces.

Section 3, Subroutines and Libraries, describes the available library functions or subroutines. Their binary versions reside in various system libraries in the directories **/lib** and **/usr/lib**. See *intro*(3) for descriptions of these libraries and the files in which they are stored.

Section 4, File Formats, documents the structure of particular kinds of files; for example, the format of the output of the link editor is given in *a.out*(4). Excluded are files used by only one command (for example, the assembler's intermediate files). In general, the C language **struct** declarations corresponding to these formats can be found in the directories **/usr/include** and **/usr/include/sys**.

Section 5, Miscellaneous Facilities, contains descriptions of character sets, macro packages, and other such information.

Section 6, Games, describes the games and educational programs that reside in the directory **/usr/games**.

Section 7, Special Files, discusses the characteristics of files that actually refer to input/output devices.

Entry organization. All entries are based on a common format, in which some parts are optional:

NAME — The **NAME** part gives the name(s) of the entry and briefly states its purpose.

SYNOPSIS — The **SYNOPSIS** part summarizes the use of the program being described. A few conventions are used, particularly in Section 1 (Commands and Application Programs):

- **Bold** — **Boldface** strings are literals, and are to be typed just as they appear.
- *Regular* — *Regular face* strings usually represent substitutable argument prototypes and program names found elsewhere in the manual.
- [ ] — Square brackets around an argument prototype indicate that the argument is optional. When an argument prototype is given as "name" or "file," it always refers to a *file* name.
- ... — Ellipses are used to show that the previous argument prototype can be repeated.
- − + = — A final convention is used by the commands themselves. An argument beginning with a minus (−), plus (+), or equal sign (=) is often taken to be some sort of flag argument, even if it appears in a position where a file name could appear. Therefore, it is unwise to have files whose names begin with −, +, or =.

DESCRIPTION — The **DESCRIPTION** part discusses the subject at hand.

EXAMPLE(S) — The **EXAMPLE(S)** part gives example(s) of usage, where appropriate.

FILES — The **FILES** part gives the file names that are built into the program.

SEEALSO — The **SEE ALSO** part gives pointers to related information.

DIAGNOSTICS — The **DIAGNOSTICS** part discusses the diagnostic indications that may be produced. Messages that are intended to be self-explanatory are not listed.

NOTES — The **NOTES** part gives information that might be helpful under the particular circumstance described.

WARNINGS — The **WARNINGS** part points out potential pitfalls.

BUGS — The **BUGS** part gives known bugs and sometimes deficiencies. Occasionally, the suggested fix is also described.

A table of contents is provided at the front of each of the four volumes, along with a complete permuted index derived from the tables. On each *index* line, the title of the entry to which that line refers is followed by the appropriate section number in parentheses. This is important because there is considerable duplication of names among the sections, arising principally from commands that exist only to exercise a particular system call.

PERMUTED INDEX

This index includes entries for all pages of Volumes 1 through 4. The entries themselves are based on the one-line descriptions or titles found in the NAME portion of each manual page; the significant words (keywords) of these descriptions are listed alphabetically down the center of the index.

The index is actually a keyword-in-context (KWIC) index that has three columns. To use the index, read the center column to look up specific commands by name or by subject topics. Note that the entry may begin in the left column or wrap around and continue into the left column. A period (.) marks the end of the entry, and a slash (/) indicates where the entry has been continued or truncated. The right column gives the manual page where the command or subject is described.

NAME

intro - introduction to file formats

DESCRIPTION

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

Entries suffixed by **(4)** describe the configuration files used with the CTIX networking packages. These files can be manipulated directly (by using a text editor) or by using *adman*(1).

NOTES

CTIX Internetworking manual pages frequently cite appropriate Requests for Comments (RFCs). RFCs can be obtained from the DDN Network Information Center, SRI International, Menlo Park, CA 94025.

SEE ALSO

CTIX Network Administrator's Guide.

NAME

a.out - common assembler and link editor output

SYNOPSIS

#include <a.out.h>

DESCRIPTION

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

A common object file consists of a file header, a CTIX system header (if the file is link editor output), a table of section headers, relocation information, (optional) line numbers, a symbol table, and a string table. The order is given below.

File header.
CTIX system header.
Section 1 header.
...
Section n header.
Section 1 data.
...
Section n data.
Section 1 relocation.
...
Section n relocation.
Section 1 line numbers.
...
Section n line numbers.
Symbol table.
String table.

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

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

When an **a.out** file is loaded into memory for execution, three logical segments are set up: the text segment, the data segment (initialized data followed by uninitialized, the latter actually being initialized to all 0's), and a stack. For files created with the **-N** or the **-z** option to *ld* (these are files with magic number 0407 and 0413, respectively), the text segment begins at location 0x80000 in the core image. For files created with the **-F** option to *ld* (these are files with magic number 0413), the text segment begins at 0x80000 plus an offset equal to the size of the headers: thus, the location of the text segment varies with the number of section headers in the *a.out* file. When the **-F** option is used to create an *a.out* file that has three sections (.text, .data, and .bss), the first address is at 0x800a8. The header is never loaded, except in the case of files with magic number 0413 created with the **-F** option to *ld*.

If the magic number is 0407, the text segment is not to be write-protected or shared; the data segment is contiguous with the text segment. If the magic number is 0413, the text segment permits demand paging and the text is not writable by the program.

Both the **-z** and **-F** options of the loader *ld*(1) create *a.out* files with magic numbers 0413. If the **-z** option is used, both the text and data sections of the file are on 1024-byte boundaries. If the **-F** option is used, the text and data sections of the file are contiguous. Loading a single 4096-byte page into memory requires 4 transfers of 1024 bytes each for **-z**, and typically one transfer of 4096 bytes for **-F**. Thus *a.out* files created with **-F** can load faster and require less disk space.

The stack begins at the end of memory and grows towards lower addresses. The stack is automatically extended as required. The data segment is extended only as requested by the *brk*(2) system call.

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

File Header

The format of the **filehdr** header is

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

CTIX System Header

The format of the CTIX system header is

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

Section Header

The format of the **section** header is

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

Relocation

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

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

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

Symbol Table

The format of each symbol in the symbol table is

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

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

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

Some symbols require more information than a single entry; they are followed by *auxiliary entries* that are the same size as a symbol entry. The format follows.

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

        struct {
                char    x_fname[FILNMLEN];
        } x_file;

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

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

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

SEE ALSO

as(1), cc(1), ld(1), brk(2), filehdr(4), ldfcn(4), linenum(4), reloc(4), scnhdr(4), syms(4).

NAME

aliases - aliases file for sendmail

SYNOPSIS

/usr/lib/aliases

DESCRIPTION

This file describes user id aliases used by the *sendmail*(1M) program. Entries are of three possible forms:

- A series of names of the form

  name: name_1, name2, name_3, . . .

  where *name* is the name to alias, and the *name_n* are the aliases for that name.

- A name with an include file specification of the form

  name: :include:filename

  where *name* is the name to alias, and *filename* is the full pathname of a file containing a list of aliases.

- A name with a pipe command specification of the form

  name: | command

  where *name* is the name to alias, and *command* is a shell command to be executed with the message as standard input.

All three forms can be combined in one entry.

If there are spaces within a name or command, it must be enclosed in double quotes.

Lines beginning with white space are continuation lines. Lines beginning with # are comments.

Aliasing occurs only on local names. Loops can not occur, since no message will be sent to any person more than once.

After aliasing has been done, local and valid recipients who have a **list of users defined in that file.**

/usr/lib/aliases is only the raw data file; the actual aliasing information is placed into a binary format in the files **/usr/lib/aliases.dir** and **/usr/lib/aliases.pag** using the program *newaliases* [see *sendmail*(1M)]. A *newaliases* command should be executed each time the aliases file is changed for the change to take effect.

SEE ALSO

sendmail(1M), dbm(3X).

BUGS

Because of restrictions in *dbm*(3X), a single alias cannot contain more than about 1000 bytes of information. You can get longer aliases by ‘‘chaining;’’ that is, make the last name in the alias be a dummy name which is a continuation alias.

NAME

ar - common archive file format

SYNOPSIS

#include <ar.h>

DESCRIPTION

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

Each archive begins with the archive magic string.

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

Each archive which contains common object files [see *a.out*(4)] includes an archive symbol table. This symbol table is used by the link editor *ld*(1) 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*.

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.

The *ar_name* field is blank-padded and slash (/) terminated. 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*(1) is used. Conversion tools such as *convert*(1) exist to aid in the transportation of non-common format archives to this format.

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

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

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

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

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

SEE ALSO

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

WARNINGS

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

NAME

cftime - language specific strings

DESCRIPTION

The programmer can create one printable file per language. These files must be kept in a special directory **/lib/cftime**. If this directory does not exist, the programmer should create it. The contents of these files are:

- abbreviated month names (in order)
- month names (in order)
- abbreviated weekday names (in order)
- weekday names (in order)
- default strings that specify formats for local time (**%x**) and local date (**%X**).
- default format for cftime, if the argument for cftime is zero or null.
- AM (ante meridian) string
- PM (post meridian) string

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 the strings appear in the file shown below.

EXAMPLE

/lib/cftime/usa_english

```
Jan
Feb
 ...
January
February
 ...
Sun
Mon
 ...
Sunday
Monday
 ...
%H:%M:%S
%m/%d/%y
%a %b %d %T %Z %Y
```

AM
PM

FILES

/lib/cftime - directory that contains the language specific printable files (create it if it does not exist)

SEE ALSO

ctime(3C).

NAME

checklist - list of file systems processed by fsck and ncheck

DESCRIPTION

checklist resides in directory **/etc** and contains a list of, at most, 15 *special file* names. Each *special file* name is contained on a separate line and corresponds to a file system. Each file system will then be automatically processed by the *fsck*(1M) command.

FILES

/etc/checklist

SEE ALSO

fsck(1M), ncheck(1M).

NAME

core - format of core image file

DESCRIPTION

CTIX writes out a core image of a terminated process when any of various errors occur. See *signal*(2) for the list of reasons; the most common errors that cause core files to be written are memory protection violations, illegal instructions, exceptions, and user-generated quit signals. The core image is called **core** and is written in the process's working directory (provided it can be; normal access controls apply). A process with an effective user ID different from the real user ID will not produce a core image.

The first section of the core image is a copy of the system's per-user data for the process, including the registers as they were at the time of the fault. The size of this section depends on the parameter **USIZE,** which is defined in **/usr/include/sys/page.h**. The remainder represents the actual contents of the user's core area when the core image was written. If the text segment is read-only and shared, or separated from data space, it is not dumped.

The format of the information in the first section is described by the *user* structure of the system, defined in **/usr/include/sys/user.h**. The important stuff not detailed therein is the locations of the registers, which are outlined in **/usr/include/sys/reg.h**.

SEE ALSO

crash(1M), sdb(1), setuid(2), signal(2).

NAME

cpio - format of cpio archive

DESCRIPTION

The *header* structure, when the **-c** option of *cpio*(1) is not used, is:

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

When the **-c** option is used, the *header* information is described by:

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

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

The last record of the *archive* always contains the name TRAILER!!!. Special files, directories, and the trailer are recorded with *h_filesize* equal to zero.

SEE ALSO

cpio(1), find(1), stat(2).

NAME

cprofile - setting up a C shell environment at login time

DESCRIPTION

cprofile is for use with *csh*(1). For every user of *csh* the system file **/etc/cprofile** is executed immediately upon login. If the user's login directory contains a file named **.cshrc**, that file will then be executed, followed by commands from the **.login** file.

The following example is typical for a user's **.cshrc** file:

```
setenv PATH :$PATH:$HOME/bin
setenv MAIL /usr/mail/myname
setenv TERM pt
```

The system file **/etc/cprofile** allows the system administrator to perform services for the entire community of *csh* users. These services include: the announcement of system news, user mail, the setting of default environmental variables, and setting the umask [see umask(1)]. In addition, **/etc/cprofile** executes special actions for the **root** login.

/etc/cprofile can be customized via four files in the **/etc/rcopts** directory:

TSETX The presence of this file overrrides the default *tset* command, and instead queries the user for terminal type with the command

```
setenv TERM `tset - '?dumb'`
```

(The default sets **TERM** to the value specified in **/etc/ttytype**.)

TPUT The presence of this file causes the execution of

```
tput init
```

which initializes the user's terminal according to the value for the **TERM** environment variable.

LOCCPRF
If this file exists, it is sourced by **/etc/cprofile**; if there are any customizations to the system **cprofile** file, they should be put in **LOCCPRF**.

AUTOWM
The presence of this file causes **wm** (window manager for Programmable Terminals and Graphics Terminals) to be *exec*ed after **.cshrc** and **.login**.

For further information about setting variables, see *csh*(1) and *sh*(1).

NOTE

Although **/etc/cprofile** is an ASCII commands text file, it is not meant to be "configurable". Configurability is provided at the level of an "rcopt", or, in the case of individual users, in **.login** and **.cshrc** files.

FILES

$HOME/.login
$HOME/.cshrc
$HOME/.logout
/etc/cprofile
/etc/rcopts/TSETX
/etc/rcopts/TPUT
/etc/rcopts/LOCCPRF
/etc/rcopts/AUTOWM

SEE ALSO

csh(1), env(1), login(1), mail(1), sh(1), stty(1), su(1), tset(1), wm(1), profile(4), ttytype(4), environ(5), term(5).
S/Series CTIX Administrator's Guide.

NAME

dir - format of directories

SYNOPSIS

#include <sys/dir.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 flag word of its i-node entry [see *fs*(4)]. The structure of a directory entry as given in the include file is:

```
#ifndef DIRSIZ
#define DIRSIZ    14
#endif
struct  direct
{
        ushort  d_ino;
        char    d_name[DIRSIZ];
};
```

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

SEE ALSO

dirent(4), fs(4).

NAME

dirent - file system independent directory entry

SYNOPSIS

#include <sys/dirent.h>
#include <sys/types.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 {
              long                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 that 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 it will always result in the next entry being on a long boundary.

FILES

/usr/include/sys/dirent.h

SEE ALSO

getdents(2).

NAME

errfile - error-log file format

SYNOPSIS

#include <sys/erec.h>

DESCRIPTION

When the system detects a hardware error, an error record is generated and passed to the error-logging daemon for recording in the error log for later analysis. The default error log is **/usr/adm/errfile**.

The format of an error record depends on the type of error that was encountered. Every record, however, has a header with the following format:

```
struct errhdr {
      short     e_type;    /* record type */
      short     e_len;     /* bytes in record (inc hdr) */
      time_t    e_time;    /* time of day */
};
```

Valid record types follow:

```
#define E_GOTS     010     /* start */
#define E_STOP     012     /* stop */
#define E_TCHG     013     /* time change */
#define E_CCHG     014     /* configuration change */
#define E_BLK      020     /* block device error */
#define E_STRAY    030     /* stray interrupt */
#define E_PRTY     031     /* memory parity */
#define E_BUSFLT   032     /* bus fault */
#define E_CONS     040     /* console string */
#define E_CONR     041     /* console record */
#define E_CONO     042     /* console overflow */
#define E_SERIAL   043     /* serial device driver error */
```

Some records in the error file are of an administrative nature. These include the startup record entered into the file when logging is activated, the stop record written if the daemon is terminated "gracefully", and the time-change record used to account for changes in the system's time-of-day. These records have the following formats:

```
struct estart {
        short           e_cpu;          /* CPU type */
        struct utsname e_name;          /* system names */
        short           e_mmr3;         /* boot reason from CDT */
        long            e_syssize;      /* system memory size */
        int             e_fhole;        /* 64K chunks of memory*/
                                        /* omitted */
        short           e_bconf;        /* block dev configuration */
        char            e_panic;        /* if reboot from panic,
                                        /* what was it */
};

#define eend errhdr     /* record header */

struct etimchg {
        time_t e_ntime; /* new time */
};
```

Stray interrupts cause a record with the following format to be logged:

```
struct estray {
        physadr         e_saddr;/* stray loc or device addr */
        short   e_sbacty;       /* active block devices */
};
```

A memory subsystem error generates the following record:

```
struct eparity {
        uint    e_gsr;  /* general status register */
};
```

Error records for block devices have the following format:

```
struct eblock {
      dev_t      e_dev;          /* "true" major + minor dev no */
      physadr    e_regloc;       /* controller address */
      short      e_bacty;        /* other block I/O activity */
      struct iostat {
        long     io_ops;         /* number read/writes */
        long     io_misc;        /* number "other" operations */
        ushort   io_unlog;       /* number unlogged errors */
      }          e_stats;
      short      e_bflags;       /* read/write, error, etc */
      short      e_trkoff;       /* logical dev start trk */
      daddr_t    e_bnum;         /* logical block number */
      ushort     e_bytes;        /* number bytes to transfer */
      paddr_t    e_memadd;       /* buffer memory address */
      ushort     e_rtry;         /* number retries */
      short      e_nreg;         /* number device registers */
      short      e_trks          /* number of heads */
      short      e_secs          /* number of physical sectors per track */
      short      e_ctlr          /* controller type */
};
```

The following values are used in the *e_bflags* word:

```
#define E_WRITE    0      /* write operation */
#define E_READ     1      /* read operation */
#define E_NOIO     02     /* no I/O pending */
#define E_PHYS     04     /* physical I/O */
#define E_MAP      010    /* Unibus map in use */
#define E_ERROR    020    /* I/O failed */
```

The error types CONS and CONO are flagged by *errdemon*(1M) and *errdead* and written to the console log **/etc/log/confile**.

A bus fault generates the following record.

```
struct ebusflt {
        short       e_type;      /* kind of fault */
        caddr_t     e_vaddr      /* virtual address of fault */
        uint        e_bsr;       /* combined bsr0 and bsr1 */
        ushort      e_pte;       /* page frame of fault */
        ushort      e_pid;       /* pid */
        uint        e_pc;        /* PC at time of fault */
        uint        e_rps;       /* RPS at time of fault */
        uint        e_regs[16];  /* all the registers */
};
```

A serial driver error generates the following reports:

```
struct      eserial {
    ushort          e_type/* type of error */
    ushort          e_dev/* which physical port */
};
```

The following types exist for e_type:

```
#define ECHLOS      0x1  /* character lost in input FIFO */
#define ERXORUN     0x2  /* receiver overrun */
#define ENOCLIST    0x4  /* no new clist available */
#define ENORBUF     0x8  /* no receive buffer available */
```

SEE ALSO

errdemon(1M).

NAME

exports - NFS file systems export configuration file

SYNOPSIS

/etc/exports

DESCRIPTION

The file */etc/exports* describes the file systems which are being exported to *NFS* clients. It is created by the system administrator using a text editor and processed by the *mount* request daemon *mountd*(1M) each time a mount request is received.

The file consists of a list of file systems and the machine names allowed to remote mount each file system. The file system names are left justified and followed by a list of names separated by white space. The names will be looked up in */etc/hosts*. Prior to granting mount requests, the hostnames of all eligible clients are expanded to include all their aliases as specified in */etc/hosts*. A file system name with no name list following means export to everyone. A "#" anywhere in the file indicates a comment extending to the end of the line it appears on. Lines beginning with white space are continuation lines.

Although the file system name can be a directory within the file system, the complete file system is actually what is exported. An NFS client can choose to mount the complete exported file system or any subdirectory within it.

EXAMPLE

```
/usr        mktg.Mysite.COM     # export to hostname
/usr/local                      # export to the world
/usr2       mktg engnode        # export to only list
```

FILES

/etc/exports

BUGS

The identification of the remote system is dependent on the local network transport mechanism employed.

SEE ALSO

mountd(1M)

NAME

filehdr - file header for common object files

SYNOPSIS

#include <filehdr.h>

DESCRIPTION

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

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

f_symptr is the byte offset into the file at which the symbol table can be found. Its value can be used as the offset in *fseek*(3S) to position an I/O stream to the symbol table. The operating system optional header is always 36 bytes. The valid magic numbers are given below.

```
#define MC68KWRMAGIC    0520    /* writeable text segments */
#define MC68KROMAGIC    0521    /* readonly shareable text segments */
#define MC68KPGMAGIC    0522    /* demand paged text segments */
```

The value in *f_timdat* is obtained from the *time*(2) system call.

Flag bits currently defined are:

```
#define  F_RELFLG   0000001    /* relocation entries stripped */
#define  F_EXEC     0000002    /* file is executable */
#define  F_LNNO     0000004    /* line numbers stripped */
#define  F_LSYMS    0000010    /* local symbols stripped */
#define  F_MINMAL   0000020    /* minimal object file */
#define  F_UPDATE   0000040    /* update file, ogen produced */
#define  F_SWABD    0000100    /* file is "pre-swabbed" */
#define  F_AR32W    0001000    /* non-DEC host, including Convergent
                               /* Technologies systems */
#define  F_PATCH    0002000    /* "patch" list in opt hdr */
```

The CPU type is encoded in bits 04000 and 010000. The FPU (floating-point unit) type is encoded in bits 0100000, 040000, and 020000. Macros are defined to set and extract the CPU and FPU values as follows:

```
SETFPU(flag, value)
SETCPU(flag, value)
GETFPU(flag)
GETCPU(flag)
```

Valid values for CPU are:

```
#define  F_M68010   0
#define  F_M68020   1
```

Valid values for FPU are:

```
#define  F_NOFPU    0
#define  F_SOFT     1
#define  F_M68881   2
#define  F_SKY      4
```

SEE ALSO

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

NAME

fs: file system - format of system volume

SYNOPSIS

#include <sys/param.h>
#include <sys/types.h>
#include <sys/filsys.h>
#include <sys/filbitmap.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.

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

```
/*
 *  Structure of the super-block.
 */
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 */
    ushort  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*/
    ushort  s_tinode;       /* total free inodes */
    char    s_fname[6];     /* file system name */
    char    s_fpack[6];     /* file system pack name */
    long    s_fill[5];      /* ADJUST to make sizeof filsys be 512 */
    struct  filbitmap *s_filbitmap;   /* in core pointer to free list bitmap */
    short   s_bfree;        /* Number of blocks free in s_filbitmap */
    short   s_bucnum;       /* Bucket currently in use */
    daddr_t s_bitaddress[4]; /* Disk addresses of buckets and bitmap */
#define S_BUCKET0       0
#define S_BUCKET1       1
```

```
#define S_BITMAP0       2
#define S_BITMAP1       3
    char    s_fsbitmap;  /* if set then file system has a valid bitmap */
    char    s_fsok;      /* fsok flag is no longer used */

    /* The following three shorts used to be used by PILF.
     * We now use them for the bitmapped free list (in core only).
     */
    short   s_singlep;   /* in core index for single block allocations */
    short   s_doublep;   /* in core index for double block allocations */
    short   s_quadp;     /* in core index for quad block allocations */

    long    s_magic;     /* magic number to indicate new file system */
    long    s_type;      /* type of new file system */
    long    s_state;     /* file system state */
    long    s_bsize;     /* file system block size */
};

#define FsMAGIC         0xfd187e20/* s_magic */

#define Fs1b            1          /* 512 byte block */
#define Fs2b            2          /* 1024 byte block */
#define Fs4b            4          /* 4096 byte block */
#define     FsPILF      0x10000    /* PILF file system */

#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 */

#define getfs(mp)       ((struct filsys *)&mp->m_bufp->b_un.b_addr[SUPERBOFF])
```

The value of *s_type* indicates the file system type. Currently, two file system types are supported: the 1024-byte logical block and the 4096-byte logical block. In the following description, a block is then determined by the type. For the original 512-byte oriented file system, a block is 512-bytes; for the 1024-byte oriented file system, a block is 1024-bytes or two sectors. for the 4096-byte oriented file system, a block is 4096-bytes or eight sectors. The operating system takes care of all conversions from logical block numbers to physical sector numbers.

If the value of *s_type* is **Fs4b**, the value of *s_bsize* determines the logical block size of the system.

The value of *s_state* indicates the state of the file system. A cleanly unmounted, undamaged file system is indicated by the FsOKAY state. After a file system has been mounted for update, the state changes to FsACTIVE. A special case is used for the **root** file system. If the **root** file system appears to be damaged at boot time, it is mounted but marked FsBAD. Lastly, after a file system has been unmounted, the state reverts to FsOKAY.

The value of *s_isize* is the address of the first logical block after the i-list; the i-list starts just after the super-block, namely in logical block 2 (sector 4); thus the i-list is *s_isize*-2 logical blocks long. The value of *s_fsize* is the first block not potentially available for allocation to a file. The system uses the values of *s_isize* and *s_fsize* to check for invalid 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 is provided on non-bitmapped file systems and 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*.

The value of *s_tfree* is the number of total free blocks available in the file system.

The value of *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 *s_ninode* 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 used 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.

The value of *s_tinode* is the number of total free i-nodes available in the file system.

The *s_flock* and *s_ilock* fields are flags maintained in the core copy of the file system while it is mounted; 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.

The *s_ronly* field is a read-only flag to indicate write-protection (in-core only).

The value of *s_time* specifies the last time the super-block of the file system was changed and the number of seconds that have elapsed since 00:00 Jan. 1, 1970 (GMT). 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.

The value of *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).

The *s_fsbitmap* flag indicates that the file system has a valid bitmap describing a number of blocks that are omitted from the free list; these blocks are placed on the bitmap (**filbitmap.h**). If this flag is set, CTIX uses the bitmap; otherwise the free list is used.

The values of *s_bitaddresses* are the disk addresses of the *filbitmap* structure. For a 1K file system, each address is for a 1024-byte logical block; for a 4K file system, each address is for a 4096-byte logical block.

All allocations of blocks are made from the bitmap. If a block being deallocated is in the section of the disk represented by *s_bucknum*, the deallocated block is put in the bitmap; if the block is not in the area represented by the bitmap, it is put on the free list.

The format of the file system bitmap and bucket list follows:

```
#define    BFLBLOCKS     16384
           /* The number of bits in the bitmap */
#define    BFLBUCKETS    1024
```

```
        /* The number of buckets in the bucket list */
#define   BFLCHARS        (BFLBLOCKS/8)
        /* The number of chars in the bitmap */
struct filbitmap {
    ushort      fb_buckets[BFLBUCKETS];
        /* list of buckets describing the free list */

    unchar      fb_bitmap[BFLCHARS];
        /* Bitmap describing free blocks not on the free list */
};
```

SEE ALSO

fsck(1M), fsdb(1M), mkfs(1M), mount(2), inode(4).

NAME

fspec - format specification in text files

DESCRIPTION

It is sometimes convenient to maintain text files on CTIX with non-standard tabs, (that is, tabs which 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 CTIX 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**,etc. The canned tabs which 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

fstab - file-system-table

DESCRIPTION

The **/etc/fstab** file contains information about file systems for use by *mount*(1M) and *mountall*(1M). Each entry in **/etc/fstab** has the following format:

column 1	block special file name of file system or advertised remote resource
column 2	mount-point directory
column 3	**-r** if to be mounted read-only; **-d**[**r**] if remote
column 4	(optional) file system type string
column 5+	ignored

White-space separates columns. Lines beginning with # are comments. Empty lines are ignored.

A file-system-table might read as follows:

```
/dev/dsk/c0d0s3 /usr S51K
/dev/dsk/c0d1s2 usr/src -r
adv_resource /mnt -d
bertha:/usr/jerry /mnt NFS
```

FILES

/etc/fstab

SEE ALSO

mount(1M), mountall(1M), rmountall(1M).

NAME

gateways - routed configuration file

DESCRIPTION

The */etc/gateways* is comprised of a series of lines, each in the following format:

< **net** | **host** > *name1* **gateways** *name2* **metric** *value* < *type* >

The **net** or **host** keyword indicates if the route is to a network or specific host.

Name1 is the name of the destination network or host. This may be a symbolic name located in */etc/networks* or */etc/hosts* [or, if started after *named*(1M), known to the name server], or an Internet address specified in "dot" notation; see *hosts*(4) and *inet*(7).

Name2 is the name or address of the gateways to which messages should be forwarded.

Value is a metric indicating the hop count to the destination host or network.

The following gateway types are defined:

Active gateways are treated equally to network interfaces. Routing information is distributed to the gateways and if no routing information is received for a period of time, the associated route is deleted.

Passive gateways are not expected to exchange routing information. They are maintained in the routing tables forever, and information regarding their existence is included in any routing information transmitted.

External gateways are also passive, but are not placed in the kernel routing table nor are they included in routing updates. The function of external entries is to inform *routed* that another routing process will install such a route, and that alternate routes to that destination should not be installed. Such entries are only required when both routers may learn of routes to the same destination.

SEE ALSO

routed(1M)

NAME

gettydefs - speed and terminal settings used by getty

DESCRIPTION

The **/etc/gettydefs** file contains information used by *getty*(1M) to set up the speed and terminal settings for a line. It supplies information on what the *login*(1) 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.

NOTE: Customers who need to support terminals that pass 8 bits to the system (as is typical outside the U.S.A.) must modify the entries in **/etc/gettydefs** as described in the **WARNINGS** section.

Each entry in **/etc/gettydefs** has the following format:

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

Each entry is followed by a blank line. The various 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 various 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*(2) settings to which the terminal is to be set if a terminal type is not specified to *getty*. The flags *getty* understands are the same as the ones listed in **/usr/include/sys/termio.h** [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*(1).

final-flags These flags take the same values as the *initial-flags* and are set just prior to *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* will search for the entry with *next-label* as its *label* field and set up the terminal for those settings. Usually, a series of speeds is 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 **/etc/gettydefs** is used, thus making the first entry of **/etc/gettydefs** the default entry. It is also used if *getty* can not find the specified *label*. If **/etc/gettydefs** itself is missing, there is one entry built into the command which will bring up a terminal at **9600** baud.

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

FILES

/etc/gettydefs

SEE ALSO

getty(1M), login(1), ioctl(2), termio(7).

WARNINGS

To support terminals that pass 8 bits to the system (also, see the BUGS section), modify the entries in the **/etc/gettydefs** file for those terminals as follows: add CS8 to *initial-flags*.

This change will permit 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*(1M) 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
```

BUGS

8-bit with parity mode is not supported.

NAME

gps - graphical primitive string, format of graphical files

DESCRIPTION

GPS is a format used to store graphical data. Several routines have been developed to edit and display GPS files on various devices. Also, higher level graphics programs such as *plot* [in *stat*(1G)] and *vtoc* [in *toc*(1G)] produce GPS format output files.

A GPS is composed of five types of graphical data or primitives.

GPS PRIMITIVES

lines The *lines* primitive has a variable number of points from which zero or more connected line segments are produced. The first point given produces a *move* to that location. (A *move* is a relocation of the graphic cursor without drawing.) Successive points produce line segments from the previous point. Parameters are available to set *color*, *weight*, and *style* (see below).

arc The *arc* primitive has a variable number of points to which a curve is fit. The first point produces a *move* to that point. If only two points are included, a line connecting the points will result; if three points a circular arc through the points is drawn; and if more than three, lines connect the points. (In the future, a spline will be fit to the points if they number greater than three.) Parameters are available to set *color*, *weight*, and *style*.

text The *text* primitive draws characters. It requires a single point which locates the center of the first character to be drawn. Parameters are *color*, *font*, *textsize*, and *textangle*.

hardware The *hardware* primitive draws hardware characters or gives control commands to a hardware device. A single point locates the beginning location of the *hardware* string.

comment A *comment* is an integer string that is included in a GPS file but causes nothing to be displayed. All GPS files begin with a comment of zero length.

GPS PARAMETERS

color *color* is an integer value set for *arc*, *lines*, and *text* primitives.

weight *weight* is an integer value set for *arc* and *lines* primitives to indicate line thickness. The value **0** is narrow weight, **1** is bold, and **2** is medium weight.

style *style* is an integer value set for *lines* and *arc* primitives to give one of the five different line styles that can be drawn on TEKTRONIX 4010 series storage tubes. They are:

0	solid
1	dotted
2	dot dashed
3	dashed
4	long dashed

font An integer value set for *text* primitives to designate the text font to be used in drawing a character string. (Currently *font* is expressed as a four-bit *weight* value followed by a four-bit *style* value.)

textsize *textsize* is an integer value used in *text* primitives to express the size of the characters to be drawn. *Textsize* represents the height of characters in absolute *universe-units* and is stored at one-fifth this value in the size-orientation (*so*) word (see below).

textangle *textangle* is a signed integer value used in *text* primitives to express rotation of the character string around the beginning point. *textangle* is expressed in degrees from the positive x-axis and can be a positive or negative value. It is stored in the size-orientation (*so*) word as a value 256/360 of it's absolute value.

ORGANIZATION

GPS primitives are organized internally as follows:

lines	*cw points sw*
arc	*cw points sw*
text	*cw point sw so* [*string*]
hardware	*cw point* [*string*]
comment	*cw* [*string*]

cw *cw* is the control word and begins all primitives. It consists of four bits that contain a primitive-type code and twelve bits that contain the word-count for that primitive.

point(s) *point*(*s*) is one or more pairs of integer coordinates. *text* and *hardware* primitives only require a single *point*. *point*(*s*) are values within a Cartesian plane or *universe* having 64K (-32K to +32K) points on each axis.

sw *sw* is the style-word and is used in *lines*, *arc*, and *text* primitives. For all three, eight bits contain *color* information. In *arc* and *lines* eight bits are divided as four bits *weight* and four bits *style*. In the *text* primitive eight bits of *sw* contain the *font*.

so — *So* is the size-orientation word used in *text* primitives. Eight bits contain text size and eight bits contain text rotation.

string — *string* is a null-terminated character string. If the string does not end on a word boundary, an additional null is added to the GPS file to insure word-boundary alignment.

SEE ALSO

graphics(1G), stat(1G), toc(1G).

NAME

group - group file

DESCRIPTION

group contains for each group the following information:

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

This is an ASCII file. The fields are separated by colons; each group is separated from the next by a new-line. If the password field is null, no password is demanded.

This file resides in directory **/etc**. Because of the encrypted passwords, it can and does have general read permission and can be used, for example, to map numerical group ID's to names.

FILES

/etc/group

SEE ALSO

newgrp(1M), passwd(1), passwd(4).

NAME

hosts - list of hosts on network

DESCRIPTION

The file **/etc/hosts** is a list of hosts that share the network, including the local host. It is referred to by programs that need to translate between host names and DARPA Internet addresses when the name server [see *named*(1M)] is not being used. Each line in the file describes a single host on the network and consists of three fields separated by any number of blanks or tabs:

address name aliases ...

where

address is the DARPA Internet address. Unless another type of address is required by some host on the network, *address* should be a Class A address, which takes the form *net.node*, where *net* is the network number from **/etc/networks** [see *networks*(4)], which must be between 0 and 127; and *node* is a value which must be unique for each host and be between 0 and 16777215.

name is the official name of the host. If the host is a computer system running CTIX, it must claim this host name by executing *hostname*(1M) when it is initializing itself.

aliases... is a list of alternate names for the host. Aliases can be used in network commands in place of the official name.

It is suggested that you specify the *hostname* and the *node name* [see *hostname*(1) and *setuname*(1M)] as aliases of one another for each machine listed in the **/etc/hosts** file.

The routines which search this file ignore comments (portions of lines beginning with #) and blank lines.

Internet addresses can actually take one of four forms:

A *A* is a simple 32-bit integer.

A.B *A* is an eight-bit quantity occupying the high-order byte and *B* is a 24-bit quantity occupying the remaining bytes. This form is suitable for a Class A address of the form *net.node*.

A.B.C — *A* is an eight-bit quantity occupying the high-order byte; *B* is an eight-bit quantity occupying the next byte; and *C* is a 16-bit quantity occupying the remaining bytes. This form is suitable for a Class B address of the form **128.***net.node.*

A.B.C.D — The four parts each occupy a byte in the address.

EXAMPLE

```
#       Engineering network

1.12    src.MySite.COM     src    net3   # Network Source Machine
1.10    test.MySite.COM    test   net2   # Network Test Machine
1.16    mifa.MySite.COM    mifa          # Software Development
1.17    mifb.MySite.COM    mifb          # Hardware Development
```

FILES

/etc/hosts
/etc/rcopts/INET-DOMAIN
/etc/rcopts/NODE

NOTE

The host lookup will be accomplished more efficiently if there is an */etc/resolv.conf* file whose contents is simply the keyword *usefile*. [See resolver(4)].

SEE ALSO

hostname(1M), setuname(1M), networks(4), inet(7).
CTIX Network Administrator's Guide.
For a discussion of network addresses, see RFC 796.

NAME

inetd.conf - configuration file for inetd (internet ''super-server'')

DESCRIPTION

inetd.conf is the configuration file for the *inetd*(1M) CTIX Internetworking ''super-server''.

The file consists of a series of single-line entries, each entry corresponding to a service to be invoked by *inetd*. These services are connection-based, datagram, or ''internal''.

Internal services are those supported by the *inetd* program: these services are ''echo'', ''discard'', ''chargen'' (character generator), ''daytime'' (human readable time), and ''time'' (machine readable time, in the form of the number of seconds since midnight, January 1, 1900). All of these services are tcp based. (For details of these services, consult the appropriate RFC from the DDN Network Information Center.)

Each service, including internal services, must have a valid entry in **/etc/services**(4). In the case of an internal service, its name must correspond to the official name of the service: that is, the first entry in **/etc/services**.

Each entry has a series of space- or tab-separated fields. (No field, except for the last one, may be omitted.) The fields are as follows:

service name
: Name of a valid service in **/etc/services**, as described above.

socket type
: One of ''stream'', ''dgram'', or ''raw'', depending on whether the socket type is stream, datagram, or raw [see *socket*(2)].

protocol
: Name of a valid protocol (for example, ''tcp'') specified in **/etc/protocols**(4).

wait/nowait
: Specifies whether the socket can be made available for new connections while there is still data waiting on the socket. The value is always ''nowait'' unless it is a datagram socket. If it is a datagram socket, the value is usually ''wait'', although ''nowait'' is possible in some cases. (Note that *tftpd* is an exception in that it must have ''wait'' specified, and yet the socket can continue to process messages on the port.)

user Name of the user as whom the server should run. This allows servers to be run with less permission than root.

server program
Except in the case of internal services, full pathname of the server program to be invoked by *inetd* when a request is waiting on a socket. For an internal service, the value is ''internal''.

server program arguments
Arguments to the server program, starting with *argv*[0], which is the name of the program. For an internal service, the value is ''internal''.

Comments are denoted by a # at the beginning of a line.

The distribution **inetd.conf** file contains prototype entries; refer to these entries when editing the file.

EXAMPLE

```
                        .
                        .
                        .
ftp      stream  tcp  nowait  root  /etc/ftpd      ftpd
telnet   stream  tcp  nowait  root  /etc/telnetd   telnetd
login    stream  tcp  nowait  root  /etc/logind    logind
exec     stream  tcp  nowait  root  /etc/execd     execd
uucpd    stream  tcp  nowait  root  /etc/uucpd     uucpd
ouucpd   stream  tcp  nowait  root  /etc/ouucpd    ouucpd
finger   stream  tcp  nowait  root  /etc/fingerd   fingerd
talk     dgram   udp  wait    root  /etc/talkd     talkd
echo     stream  tcp  nowait  root  internal
discard  stream  tcp  nowait  root  internal
chargen  stream  tcp  nowait  root  internal
daytime  stream  tcp  nowait  root  internal
time     stream  tcp  nowait  root  internal
echo     dgram   udp  wait    root  internal
discard  dgram   udp  wait    root  internal
chargen  dgram   udp  wait    root  internal
daytime  dgram   udp  wait    root  internal
time     dgram   udp  wait    root  internal
                        .
                        .
                        .
```

SEE ALSO

fingerd(1M), ftpd(1NM, inetd(1M), rexecd(1M), rlogind(1M), rshd(1M), talkd(1M), telnetd(1M), tftpd(1M), uucpd(1M), protocols(4), services(4).
CTIX Network Administrator's Guide.

NAME

inittab - script for the init process

DESCRIPTION

The *inittab* file supplies the script to *init*'s role as a general process dispatcher. The process that constitutes the majority of *init*'s process dispatching activities is the line process **/etc/getty** that initiates individual terminal lines. Other processes typically dispatched by *init* are daemons and the shell.

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 can be inserted in the *process* field by using the *sh*(1) convention for comments. Comments for lines that spawn *gettys* are displayed by the *who*(1) command; they are expected to contain some information about the line, such as the location. There are no limits (other than maximum entry size) imposed on the number of entries within the *inittab* file. The entry fields follow:

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

rstate Defines the *run-level* in which this entry is to be processed. Run-levels effectively correspond to a configuration of processes in the system: 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-level, 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 20-second grace period before being forcibly terminated by a 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 to 6. If no run-level is specified, the process is assumed to be valid at all run-levels 0 through 6. Three other values, **a**, **b**, and **c**, can appear in the *rstate* field, even though they are not true run-levels. Entries with these characters in the *rstate* field are processed only when the *telinit* [see *init*(1M)] process requests them to be run (regardless of the system's current run-level). Note 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. Such processes are killed only if their line in **/etc/inittab** is marked **off** in the *action* field, their line is deleted entirely from **/etc/inittab**, or *init* goes into the *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, start the process; do not wait for its termination (continue scanning the *inittab* file), and when it dies restart the process. If the process currently exists, do nothing and continue scanning the *inittab* file.

wait Upon *init*'s entering 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 Upon *init*'s entering a run-level that matches the entry's *rstate*, start the process; do not wait for its termination. When the process dies, do not restart the process. If upon entering a new run-level, where the process is still running from a previous run-level change, the program is not restarted.

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

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

powerfail Execute the process associated with this entry only when *init* receives a power fail signal [**SIGPWR**; see *signal*(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 20 seconds before forcibly terminating the process by using 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 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. If this entry exists, *init* uses it to determine which run-level to enter initially; *init* uses the highest run-level specified in the **rstate** field as its initial state. If the *rstate* field is empty, this is interpreted as **0123456**, so *init* enters run-level **6**. Additionally, if *init* does not find an **initdefault** entry in **/etc/inittab**, it requests an initial run-level from the user at reboot time.

sysinit Entries of this type are executed before *init* tries to access the console. It is expected that this entry will be used only to initialize devices on which *init* might try to ask the run-level question. These entries are executed and waited for before continuing.

process A *sh* 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. Comments can be inserted with the **;** *#comment* syntax.

FILES

/etc/inittab

SEE ALSO

getty(1M), init(1M), sh(1), who(1), exec(2), open(2), signal(2).
S/Series CTIX Administrator's Guide.

NAME

inode - format of an i-node

SYNOPSIS

#include <sys/types.h>
#include <sys/ino.h>

DESCRIPTION

An i-node for a plain file or directory in a file system has the following structure defined by **<sys/ino.h>**.

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

struct      dinode
{
      ushort  di_mode;        /* mode and type of file */
      short   di_nlink;       /* number of links to file */
      ushort  di_uid;         /* owner's user id */
      ushort  di_gid;         /* owner's group id */
      off_t   di_size;        /* number of bytes in file */
      char    di_addr[39];    /* disk block addresses */
      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 created */
};

/* The 39 address bytes: 13 addresses of 3 bytes.
 *    The 40th byte is used as a generation count
 *    to detect the disk inode being reused. */
```

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

SEE ALSO

stat(2), fs(4), types(5).

NAME

issue - issue identification file

DESCRIPTION

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

FILES

/etc/issue

SEE ALSO

login(1).

NAME

ldfcn - common object file access routines

SYNOPSIS

#include <stdio.h>
#include <filehdr.h>
#include <ldfcn.h>

DESCRIPTION

The common object file access routines are a collection of functions for reading common object files and archives containing common object files. Although the calling program must know the detailed structure of the parts of the object file that it processes, the routines effectively insulate the calling program from knowledge of the overall structure of the object file.

The interface between the calling program and the object file access routines is based on the defined type **LDFILE**, defined as **struct ldfile**, declared in the header file **ldfcn.h**. The primary purpose of this structure is to provide uniform access to both simple object files and to object files that are members of an archive file.

The function *ldopen*(3X) allocates and initializes the **LDFILE** structure and returns a pointer to the structure to the calling program. The fields of the **LDFILE** structure may be accessed individually through macros defined in **ldfcn.h** and contain the following information:

LDFILE	*ldptr;
TYPE(ldptr)	The file magic number used to distinguish between archive members and simple object files.
IOPTR(ldptr)	The file pointer returned by *fopen* and used by the standard input/output functions.
OFFSET(ldptr)	The file address of the beginning of the object file; the offset is non-zero if the object file is a member of an archive file.
HEADER(ldptr)	The file header structure of the object file.

The object file access functions themselves may be divided into four categories:

(1) functions that open or close an object file

ldopen(3X) and *ldaopen* [see *ldopen*(3X)]
open a common object file

ldclose(3X) and *ldaclose* [see *ldclose*(3X)]
close a common object file

(2) functions that read header or symbol table information

ldahread(3X)
read the archive header of a member of an archive file

ldfhread(3X)
read the file header of a common object file

ldshread(3X) and *ldnshread* [see *ldshread*(3X)]
read a section header of a common object file

ldtbread(3X)
read a symbol table entry of a common object file

ldgetname(3X)
retrieve a symbol name from a symbol table entry or from the string table

(3) functions that position an object file at (seek to) the start of the section, relocation, or line number information for a particular section.

ldohseek(3X)
seek to the optional file header of a common object file

ldsseek(3X) and *ldnsseek* [see *ldsseek*(3X)]
seek to a section of a common object file

ldrseek(3X) and *ldnrseek* [see *ldrseek*(3X)]
seek to the relocation information for a section of a common object file

ldlseek(3X) and *ldnlseek* [see *ldlseek*(3X)]
seek to the line number information for a section of a common object file

ldtbseek(3X)
seek to the symbol table of a common object file

(4) the function *ldtbindex*(3X), which returns the index of a particular common object file symbol table entry.

These functions are described in detail on their respective manual pages.

All the functions except *ldopen*(3X), *ldgetname*(3X), *ldtbindex*(3X) return either **SUCCESS** or **FAILURE**, both constants defined in **ldfcn.h**. *Ldopen*(3X) and *ldaopen* [(see *ldopen*(3X)] both return pointers to an **LDFILE** structure.

Additional access to an object file is provided through a set of macros defined in **ldfcn.h**. These macros parallel the standard input/output file reading and manipulating functions, translating a reference of the **LDFILE** structure into a reference to its file descriptor field.

The following macros are provided:

GETC(ldptr)
FGETC(ldptr)
GETW(ldptr)
UNGETC(c, ldptr)
FGETS(s, n, ldptr)
FREAD((char *) ptr, sizeof (*ptr), nitems, ldptr)
FSEEK(ldptr, offset, ptrname)
FTELL(ldptr)
REWIND(ldptr)
FEOF(ldptr)
FERROR(ldptr)
FILENO(ldptr)
SETBUF(ldptr, buf)
STROFFSET(ldptr)

The STROFFSET macro calculates the address of the string table. See the manual entries for the corresponding standard input/output library functions for details on the use of the rest of the macros.

The program must be loaded with the object file access routine library **libld.a**.

SEE ALSO

fseek(3S), ldahread(3X), ldclose(3X), ldfhread(3X), ldgetname(3X), ldlread(3X), ldlseek(3X), ldohseek(3X), ldopen(3X), ldrseek(3X), ldshread(3X), ldtbindex(3X), ldtbread(3X), ldtbseek(3X), stdio(3S), intro(5).

WARNING

The macro **FSEEK** defined in the header file **ldfcn.h** translates into a call to the standard input/output function *fseek*(3S). **FSEEK** should not be used to seek from the end of an archive file since the end of an archive file may not be the same as the end of one of its object file members!

NAME

limits - file header for implementation-specific constants

SYNOPSIS

#include <limits.h>

DESCRIPTION

The header file *<limits.h>* is a list of magnitude limitations imposed by a specific implementation of the operating system. All values are specified in decimal.

```
#define ARG_MAX   10240          /* max length of arguments to exec */
#define CHAR_BIT  8              /* # of bits in a "char" */
#define CHAR_MAX  127            /* max integer value of a "char" */
#define CHAR_MIN  -128           /* min integer value of a "char" */
#define CHILD_MAX 25             /* max # of processes per user id */
#define CLK_TCK   60             /* # of clock ticks per second */
#define DBL_DIG   16             /* digits of precision of a "double" */
#define DBL_MAX   1.79769313486231470e+308 /* max decimal value of
                                 a "double" */
#define DBL_MIN   4.94065645841246544e-324  /* min decimal value of
                                 a "double" */
#define FCHR_MAX  1048576        /* max size of a file in bytes */
#define FLT_DIG   7              /* digits of precision of a "float" */
#define FLT_MAX   3.40282346638528860e+38 /* max decimal value of
                                 a "float" */
#define FLT_MIN   1.40129846432481707e-45  /* min decimal value of
                                 a "float" */
#define HUGE_VAL  3.40282346638528860e+38 /*error value returned
                                 by Math lib*/
#define INT_MAX   2147483647     /* max decimal value of an "int" */
#define INT_MIN   -2147483648    /* min decimal value of an "int" */
#define LINK_MAX  1000           /* max # of links to a single file */
#define LONG_MAX  2147483647     /* max decimal value of a "long" */
#define LONG_MIN  -2147483648    /* min decimal value of a "long" */
#define NAME_MAX  14             /* max # of characters in a file name */
#define OPEN_MAX  20             /* max # of files a process can have
                                 open */
#define PASS_MAX  8              /* max # of characters in a password */
#define PATH_MAX  256            /* max # of characters in a path name */
#define PID_MAX   30000          /* max value for a process ID */
#define PIPE_BUF  9216           /* max # bytes atomic in write to a
                                 pipe */
```

```
#define PIPE_MAX    9216         /* max # bytes written to a pipe in a
                                    write */
#define SHRT_MAX    32767        /* max decimal value of a "short" */
#define SHRT_MIN    -32767       /* min decimal value of a "short" */
#define STD_BLK     1024         /* # bytes in a physical I/O block */
#define SYS_NMLN    9            /* # of chars in uname-returned
                                    strings */
#define UID_MAX     30000        /* max value for a user or group ID */
#define USI_MAX     4294967296   /* max decimal value of an
                                    "unsigned" */
#define WORD_BIT    32           /* # of bits in a "word" or "int" */
```

WARNING

Three of these parameters are tuneable: **CHILDMAX**, **FCHR_MAX**, and **OPEN_MAX**. Their values can be changed either by reconfiguring the kernel or by running *uconf*(1M).

NAME

linenum - line number entries in a common object file

SYNOPSIS

#include <linenum.h>

DESCRIPTION

The *cc* command generates an entry in the object file for each C source line on which a breakpoint is possible [when invoked with the **-g** option; see *cc*(1)]. Users can then refer to line numbers when using the appropriate software test system [see *sdb*(1)]. The structure of these line number entries appears below.

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

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

```
l_addr                  l_lnno

function symtab index   0
physical address        line
physical address        line
...
function symtab index   0
physical address        line
physical address        line
...
```

SEE ALSO

cc(1), sdb(1), a.out(4).

NAME

/usr/adm/loginlog - log of failed login attempts

DESCRIPTION

After five unsuccessful login attempts, all the attempts are logged in the **loginlog** file. This file contains one record for each failed attempt. Each record contains the following information:

login name
tty specification
time

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

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

FILES

/usr/adm/loginlog

SEE ALSO

login(1), passwd(1), passwd(1).

NAME

master - master device information table

DESCRIPTION

The */etc/master* file is used by the *config*(1M) program to obtain device information to generate the configuration files. Do *not* modify the *master* file unless you *fully* understand its construction. The file consists of four parts, each separated by a line with a dollar sign ($) in column 1. Part 1 contains device information; part 2 contains loadable module dependencies; part 3 contains names of devices that have aliases; part 4 contains tunable parameter information. Any line with an asterisk (*) in column 1 is treated as a comment.

Part 1 contains one-line entries of 7 or 10 fields, with the fields delimited by tabs and/or blanks:

Field 1: Device name (8 chars. maximum).

Field 2: Device mask. This can be specified in octal or as a string of uppercase characters; the character corresponding to the octal value of each flag is shown in parentheses after the octal value. Each ''on'' bit indicates that the handler exists:

001000	(E)	has release handler for downloadable drivers
000200	(T)	tty header exists
000100	(N)	initialization handler
000040	(P)	power-failure handler
000020	(O)	open handler
000010	(C)	close handler
000004	(R)	read handler
000002	(W)	write handler
000001	(I)	ioctl handler.

For a file system type, field 2 is an octal mask of the presence/absence of the 32 entries in the file system switch for this particular file system type.

Field 3: Device type indicator. This can be specified in octal or as a string of lowercase characters; the character corresponding to the octal value of each flag is shown in parentheses after the octal value:

0100000	(q)	Module depends on another module.
0400000	(z)	Supply major/minor to driver, else just minor.
0200000	(d)	Line discipline.
0100000	(f)	Framework/stream type device.
0040000	(m)	Framework/stream module.

0020000	(a)	Generate xx_addr array entry.
0010000	(s)	Software module.
0004000	(x)	Not a driver; configurable module.
0002000	(j)	File system type.
0001000	(u)	Cluster device.
0000400	(v)	VME device - obsolete, do not use.
0000200	(o)	Allow only one of these devices.
0000100	(n)	Suppress device count field.
0000040	(p)	Suppress interrupt vector.
0000020	(r)	Required device.
0000010	(b)	Block device.
0000004	(c)	Character device.
0000002	(l)	Floating vector.
0000001	(i)	Fixed vector.

Field 4: Handler prefix (four characters maximum).

Field 5: Major device number for block-type device.

Field 6: Major device number for character-type device.

Field 7: Maximum number of devices on system.

Field 8: Device vector size.

Field 9: Device address type (VME modifier).

Field 10: Device interrupt level.

Part 2 of the *master* file contains any dependency specifications. If a module has the dependency flag set (in field 3 of part 1 of the *master* file), the dependency must be defined here. A dependency entry is one line, consisting of the dependent driver's name, an equal sign (=), and the name of the driver on which the driver depends:

dependent_driver=driver

Part 3 of the *master* file contains one-line entries with two fields each:

Field 1: Alias name of the device (eight characters maximum).

Field 2: Reference name of device (eight characters maximum; specified in part 1).

Part 3 contains one-line entries with two or three fields each:

Field 1: Parameter name (as it appears in description file); 20 characters maximum.

Field 2: Parameter name (as it appears in the **conf.c** file); 20 characters maximum.

Field 3: Default parameter value (20 characters maximum); parameter specification is required if this field is omitted. Some parameters, if specified as zero, are dynamically sized by the kernel at boot time.

FILES

/etc/master

SEE ALSO

config(1M), uconf(1M).

NAME

mnttab - mounted file system table

SYNOPSIS

#include <mnttab.h>

DESCRIPTION

The *mnttab* file resides in directory **/etc** and contains a table of devices, mounted by the *mount*(1M) command, in the following structure as defined by **<mnttab.h>**:

```
#define MNTNM 32
#define MNTTYP 16
#define MNTOPTS 64
struct mnttab {
        char    mt_dev[MNTNM],
                mt_filsys[MNTNM];
        short   mt_ro_flg;
        time_t  mt_time;
        char    mt_fstyp[MNTTYP];
        char    mt_mntopts[MNTOPTS];
};
```

Each entry is 150 bytes in length; the first 32 bytes are the null-padded name of the directory where the *special file* is mounted; the next 32 bytes represent the null-padded root name of the mounted special file; the next 6 bytes contain the mounted *special file*'s read/write permissions and the date on which it was mounted; the following 16 bytes are the null-padded name of the file system type; and the remaining 64 bytes are the null-padded string of mount options. Both file system type and mount options can be null strings. The mount options are used only in the case of an NFS file system.

The maximum number of entries in *mnttab* is based on the system parameter NMOUNT located in **/etc/master**, which defines the number of mounted special files.

SEE ALSO

mount(1M), setmnt(1M).

NAME

netcf - Network Configuration File

DESCRIPTION

/etc/netcf describes the structure of the available networking protocols and interfaces. It currently supports three levels of interface: Transport (TLI), Link (LLC1), and Sockets using the BSD compatibility module. The file is typically used at boot time to configure the streams drivers used for networking into the linked configuration used while running, and to initialize the BSD compatability module (socket stream head).

/etc/netcf consists of several sections describing different elements of the network configuration. These sections are meant to be modifiable by automatic installation and update programs. Each section begins with the appropriate keyword prefixed by an **!** (for example, *!section-name)* at the start of a line. An asterisk indicates that the rest of the line is a comment and should be ignored.

The TRANSPORT section describes the possible transport providers and the support protocols they need above link level. If a protocol only runs over a subset of the interfaces, that should be noted with an only keyword statement.

The format of each line is

Provider [support...] [only: if, if ...]

For example:

```
!TRANSPORT
tcp     ip            * Transmission Control Protocol
udp     ip            * User Datagram Protocol
arp     only: enet    * Address Resolution Protocol (not TLI!)
```

The INTERFACE section describes the link level interfaces available to the networking system. Each consists of a interface driver, a name, a device that supports it, convergence modules required to connect various higher level protocols, and flags. Each interface is assumed to support arbitrary higher level protocols with the LLI interface unless the only keyword is used in place of a protocol in the convergence specification.

The flags are:

S Single unit only.

M Multiple units, select with minor device number.

U Multiple units, use UNIT_SELECT *ioctl*(2) to select. (Necessary for multi-protocol, multi-unit devices.)

D Dynamic: not linked in at boot time. (Used for switched serial links.)

The format of each line is

Interface Name Device (proto: convergence..., ...) flags

For example:

```
!INTERFACE
enet      en   /dev/enet      (ip: arpproc)   U
llcloop   lo   /dev/llcloop                   S
slip      sl   /dev/slip                      UD
```

The DEVICES section creates a mapping between the driver name and the device name in the file system. If the stream entity is a stream module instead of a stream driver, the keyword module is used instead of the file name.

The format of each line is

Driver Filename

For example:

```
!DEVICES
tc        /dev/inet/tcp
ip        /dev/inet/ip
arp       /dev/inet/arp
arpproc   module
```

The SUBDRIVER section describes which drivers should be loaded together. Typically these drivers are all in the same object module.

The format of each line is

Primary [Secondary ...]

For example:

```
!SUBDRIVER
ip        icmp
arp       arpproc
```

The SOCKET section describes protocols that are to be accessed via the sockets compatibilty driver. It describes the family, type, and protocol number of the protocol to use and also has a set of flags describing the behavior of the protocol. Families and types can be specified using mnemonics. The currently defined set includes:

Families: INET (internetwork: UDP, TCP, etc.), UNSPEC (unspecified)

Types: STREAM (stream socket), DGRAM (datagram socket), RAW (raw-protocol)

The following flags are defined:

M This protocol supports atomic messages only.

C Connections are required.

A Messages contain addresses.

R Rights can be passed with this protocol.

The format of each line is:

Family Type ProtoNum Flags Protocol

For example:

```
!SOCKET
INET    STREAM  6    C    tcp
INET    DGRAM   17   AM   udp
INET    RAW          AM   icmp
```

FILES

/etc/netcf

SEE ALSO

slink(1), intro(7), ioctl(2).
CTIX Network Administrator's Guide.

NAME

netrc - login file for remote networks

DESCRIPTION

If the **.netrc** file exists, it will be used by *ftp*(1) for automatic login on the remote host. For each remote host, the file contains a one-line entry that describes the login data for the user on that host.

An entry may consist of up to three blank-separated fields introduced by keywords. The keyword is followed by the literal data needed for login. The following keywords are available:

machine The hostname of the machine.

login The user login name for that host.

password (Optional) The user's password on that host. **NOTE:** The literal password must be given in clear text; it is not encrypted.

If the **.netrc** file includes the password feature, permissions on the file must be set to prohibit reading by group and others; the file will not otherwise take effect.

EXAMPLE

The following example entry allows automatic login on the **mynode.Mysite.COM** host by a user named **myname** whose password is **kebs#1**.

```
machine mynode.Mysite.COM login myname passwd kebs#1
```

FILES

$HOME/.netrc

SEE ALSO

ftp(1).
CTIX Network Administrator's Guide.

WARNING

For security reasons, use of the password feature is not recommended.

NAME

networks - names and numbers for the internet

DESCRIPTION

The file **/etc/networks** lists networks on the internet. Each line describes a single network and consists of the following blank separated fields:

name number aliases ...

where

name is the official name of the network. All hosts on the internet should use the same official name for a given network.

number is the network number, which serves as part of the DARPA Internet address for each host on the internet. All hosts on the internet must use the same number for a given network.

aliases ... is a blank-separated list of local aliases for the network.

The routines that search this file ignore comments (portions of lines beginning with #) and blank lines.

EXAMPLE

```
# Building 1 Internet
Engineering 1    #R&D
Production 2     #Administration, etc.
```

SEE ALSO

hosts(4).
CTIX Network Administrator's Guide.

FILES

/etc/networks

NAME

passwd - password file

DESCRIPTION

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

login name
encrypted password
numerical user ID
numerical group ID
user name
initial working directory
program to use as shell

This is an ASCII file. Each field within each user entry is separated from the next by a colon. Each user entry is separated from the next by a newline. If the password field is null, no password is demanded; if the shell field is null, **/bin/sh** is used.

The file contains user login information; it has general read permission and can be used, for example, to map numerical user IDs to names.

Note that if an **/etc/shadow** file exists, encrypted passwords are stored in the **/etc/shadow** file, not in **/etc/passwd**. The password field remains in **/etc/passwd** for compatibility reasons only when **/etc/shadow** exists. If the password field in **/etc/passwd** contains an **x**, the encrypted password for that login is stored in the **/etc/shadow** file. If the login does not have a password, the password field in **/etc/passwd** is empty.

If **/etc/shadow** does not exist and the login has a password, the password field in **/etc/passwd** contains the encrypted password.

The encrypted password consists of 13 characters chosen from a 64-character alphabet (**.**, **/**, **0-9**, **A-Z**, **a-z**), except when the password is null, in which case the encrypted password is also null. Password aging is in effect for a user if the encrypted password is followed by a comma and a non-null string of characters from the above alphabet. (Such a string must be introduced in the first instance by the super-user.)

The first character of the age, *M* say, denotes the maximum number of weeks for which a password is valid. A user who attempts to log in after the password has expired is forced to supply a new one. The next character, *m* say, denotes the minimum period in weeks which must expire before the password can be changed. The remaining characters define the week (counted from the beginning of 1970) when the password was last changed. (A null string is equivalent to zero.) *M* and *m* have numerical values in the range 0-63 that correspond to the 64-character alphabet shown above (for example, / = 1 week; z = 63 weeks). If $m = M = 0$ (derived from the string . or ..), the user must change the password at the next login (and the "age" disappears from the password file entry). If $m > M$ (signified by the string ./), only the super-user can change the password.

FILES

/etc/passwd
/etc/shadow
/etc/opasswd
/etc/oshadow

SEE ALSO

login(1), passwd(1), passmgmt(1M), a64l(3C), getpwent(3C), getspent(3X), group(4), shadow(4).

NAME

plot - graphics interface

DESCRIPTION

Files of this format are produced by routines described in *plot*(3X) and are interpreted for various devices by commands described in *tplot*(1G). A graphics file is a stream of plotting instructions. Each instruction consists of an ASCII letter usually followed by bytes of binary information. The instructions are executed in order. A point is designated by four bytes representing the **x** and **y** values; each value is a signed integer. The last designated point in an **l**, **m**, **n**, or **p** instruction becomes the "current point" for the next instruction.

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

m move: The next four bytes give a new current point.

n cont: Draw a line from the current point to the point given by the next four bytes [see *tplot*(1G)].

p point: Plot the point given by the next four bytes.

l line: Draw a line from the point given by the next four bytes to the point given by the following four bytes.

t label: Place the following ASCII string so that its first character falls on the current point. The string is terminated by a newline.

e erase: Start another frame of output.

f linemod: Take the following string, up to a newline, as the style for drawing further lines. The styles are "dotted", "solid", "longdashed", "shortdashed", and "dotdashed". Effective only for the **-T4014** and **-Tver** options of *tplot*(1G) (TEKTRONIX 4014 terminal and Versatec plotter).

s space: The next four bytes give the lower left corner of the plotting area; the following four give the upper right corner. The plot will be magnified or reduced to fit the device as closely as possible.

Space settings that exactly fill the plotting area with unity scaling appear below for devices supported by the filters of *tplot*(1G). The upper limit is just outside the plotting area. In every case the plotting area is taken to be square; points outside may be displayable on devices whose face is not square.

DASI 300	space(0, 0, 4096, 4096);
DASI 300s	space(0, 0, 4096, 4096);

DASI 450	space(0, 0, 4096, 4096);
TEKTRONIX 4014	space(0, 0, 3120, 3120);
Versatec plotter	space(0, 0, 2048, 2048);

SEE ALSO

graph(1G), tplot(1G), plot(3X), gps(4), term(5).

WARNING

The plotting library *plot*(3X) and the curses library *curses*(3X) both use the names erase( ) and move( ). The curses versions are macros. If you need both libraries, put the *plot*(3X) code in a different source file than the *curses*(3X) code, and/or #undef move( ) and erase( ) in the *plot*(3X) code.

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. These services include: the announcement of system news, user mail, the setting of default environmental variables, setting the umask [see umask(1)], and the execution of **/etc/TIMEZONE** [see timezone(4)]. In addition, **/etc/profile** executes special actions for the **root** login.

The system file **/etc/profile** can be customized via four files in the **/etc/rcopts** directory:

TSETX The presence of this file overrides the default *tset* command, and instead queries the user for terminal type with the command

```
TERM = `tset - '?dumb``
export TERM
```

(The default sets **TERM** to the value specified in **/etc/ttytype**.)

TPUT The presence of this file causes the execution of

```
tput init
```

which initializes the user's terminal according to the value for the **TERM** environment variable.

LOCPRF

If this file exists, it is executed by **/etc/profile**; if there are any customizations to the system **profile** file, they should be put in **LOCPRF**.

AUTOWM

The presence of this file causes **wm** (window manager for Programmable Terminals and Graphics Terminals) to be *exec*'ed after **.profile**.

The file **$HOME/.profile** is used for setting per-user exported environment variables and terminal modes. The following example is typical for a user's **.profile** file:

```
PATH=:$PATH:$HOME/bin
MAIL=/usr/mail/myname
TERM=pt
export PATH MAIL TERM
```

FILES

/etc/TIMEZONE	timezone environment
$HOME/.profile	user-specific environment
/etc/profile	system-wide environment

SEE ALSO

env(1), login(1), mail(1), sh(1), stty(1), su(1), tput(1), cprofile(4), terminfo(4), timezone(4), environ(5), term(5).
S/Series CTIX Administrator's Guide.

NOTES

Although **/etc/profile** is an ASCII commands text file, it is not meant to be "configurable". Configurability is provided at the level of "rcopts", or, in the case of individual users, in **.profile** files.

NAME

protocols - list of Internet protocols

DESCRIPTION

The file **/etc/protocols** lists known DARPA Internet protocols. Each line describes a single protocol and consists of the following blank separated fields:

name number aliases ...

where

name is the official name of the protocol.

number is the protocol number.

aliases ... is a blank-separated list of local aliases for the protocol.

The routines that search this file ignore comments (portions of lines beginning with #) and blank lines.

Protocol names and numbers are specified by the DDN Network Information Center. Do not change this file.

FILES

/etc/protocols

SEE ALSO

CTIX Network Administrator's Guide.

NAME

queuedefs - at/batch/cron queue description file

SYNOPSIS

/usr/lib/cron/queuedefs

DESCRIPTION

The *queuedefs* file describes the characteristics of the queues managed by *cron*(1M). Each non-comment line in this file describes one queue, in the following format:

q.[*njob***j**][*nice***n**][*nwait***w**]

where

q Is the name of the queue. **a** is the default queue for jobs started by *at*(1); **b** is the default queue for jobs started by *batch*(1); **c** is the default queue for jobs run from a **crontab** file.

njob The maximum number of jobs that can be run simultaneously in that queue; if more than *njob* jobs are ready to run, only the first *njob* jobs are run, and any others are run as jobs terminate. The default value is 100.

nice The *nice*(1) value to give to all jobs in that queue that are not run with a user ID of super-user. The default value is 2.

nwait The number of seconds to wait before rescheduling a job that was deferred because more than *njob* jobs were running in that job's queue, or because more than 25 jobs were running in all the queues. The default value is 60.

Lines beginning with # are comments, and are ignored.

EXAMPLE

a.4j1n
b.2j2n90w

This file specifies that the **a** queue, for *at* jobs, can have up to four jobs running simultaneously; those jobs will be run with a *nice* value of 1. As no *nwait* value was given, if a job cannot be run because too many other jobs are running, *cron* waits 60 seconds before trying again to run it. The **b** queue, for *batch* jobs, can have up to two jobs running simultaneously; those jobs are run with a *nice* value of 2. If a job cannot be run because too many other jobs are running, *cron* waits 90 seconds before trying again to run it. All other queues can have up to 100 jobs running simultaneously; they are run with a *nice* value of 2, and if a job cannot be run because too many other jobs are running, *cron* waits 60 seconds before trying again to run it.

FILES

/usr/lib/cron/queuedefs

SEE ALSO

cron(1M), at(1).

NAME

reloc - relocation information for a common object file

SYNOPSIS

#include <reloc.h>

DESCRIPTION

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

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

#define R_ABS     0

/*
 *Motorola Processors 68000, 68010, and 68020
 *
 *
 */
#define R_RELBYTE        017
#define R_RELWORD        020
#define R_RELLONG        021
#define R_PCRBYTE        022
#define R_PCRWORD        023
#define R_PCRLONG        024
```

As the link editor reads each input section and performs relocation, the relocation entries are read. They direct how references found within the input section are treated.

R_ABS The reference is absolute and no relocation is necessary. The entry will be ignored.

R_RELBYTE A direct 8-bit reference to the symbol's virtual address.

R_RELWORD A direct 16-bit reference to the symbol's virtual address.

R_RELLONG A direct 32-bit reference to the symbol's virtual address.

R_PCRBYTE A ''PC-relative'' 8-bit reference to the symbol's virtual address. The actual address is calculated by adding a constant to the PC value.

R_PCRWORD A "PC-relative" 16-bit reference to the symbol's virtual address. The actual address is calculated by adding a constant to the PC value.

R_PCRLONG A "PC-relative" 32-bit reference to the symbol's virtual address. The actual address is calculated by adding a constant to the PC value.

More relocation types exist for other processors. Equivalent relocation types on different processors have equal values and meanings. New relocation types will be defined (with new values) as they are needed.

Relocation entries are generated automatically by the assembler and automatically used by the link editor. Link editor options exist for both preserving and removing the relocation entries from object files.

SEE ALSO

as(1), ld(1), a.out(4), syms(4).

NAME

resolv.conf - resolver configuration file

SYNOPSIS

/etc/resolv.conf

DESCRIPTION

The resolver configuration file contains information that is read by the resolver routines the first time they are invoked by a process. The file contains a list of name-value pairs that provides various types of resolver information.

This file is necessary only on a machine that will run networking programs that use the Internet Domain name server, but will not actually run the name server locally [see *named*(1M)].

The different configuration options are:

nameserver
: followed by 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; if more than one name server is specified, the resolver library queries each one in the order listed. If no *nameserver* entries are present, the default is to use the name server on the local machine. 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. (It is recommended that this file *not* be present on a machine running the name server.)

domain followed by an Internet domain name, that is the default domain to append to names that do not have a dot in them. If no *domain* entries are present, the domain returned by *gethostname*(2) is used (everything after the first '.'). Finally, if the host name does not contain a domain part, the root domain is assumed. [See *resolver*(3) for information regarding the search scheme used by resolver routines.]

usefile If this option is present, no attempt is made to contact a name server, and the **/etc/hosts** file is used.

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

EXAMPLE

```
domain       MySite.COM
nameserver   3.0.0.18
nameserver   3.0.0.14
```

FILES

/etc/resolv.conf

SEE ALSO

named(1M), gethostbyname(3), resolver(3), hosts(4), inet(7).

NAME

rfmaster - Remote File Sharing name server master file

DESCRIPTION

The **rfmaster** file is an ASCII file that identifies the nodes that are responsible for providing primary and secondary domain name service for Remote File Sharing domains. This file 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:

p The **p** type defines the primary domain name server. For this type, *name* is the domain name and *data* is the full node name of the machine that is the primary name server. The full node 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 through which the previously mentioned name servers can be reached. *Name* is the full domain name for the machine and *data* is the network address of the "listener" service on that machine [see *nlsadmin* (1M)].

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. There should also be at least one secondary name server in each domain.

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

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.

NOTE: It is highly recommended that *adman*(1M) be used to maintain/update the rfmaster file.

EXAMPLE

An example of an **rfmaster** file for domain **du** over an *Internet* transport provider is shown below. In this example, the node engnode has an internet address of 3.180.0.7 and the node mktnode has an internet address of 3.180.0.5.

```
du           p    du.engnode
du           s    du.mktnode
du.engnode   a    \x0002040103b40007
du.mktnode   a    \x0002040103b40005
```

NOTE: If a line in the **rfmaster** file begins with a # character, the entire line will be treated as a comment.

FILES

/usr/nserve/rfmaster

SEE ALSO

rfstart(1M), getservaddr(1M), nlsadmin(1M), hosts(4), services(4).
S/Series CTIX Administrator's Guide.

NAME

rhosts - remote equivalent users

DESCRIPTION

These files grant permission for remote users to use local user names without knowing the corresponding user passwords. This is known as making the remote user "equivalent" to the local user and is convenient, for example, when one person owns user names on more than one host.

If a user's home directory contains a file named **.rhosts**, remote users specified in the file are equivalent to the local user. Each user specification in the file consists of the remote user host name and user name, separated by a space. (If an asterisk is substituted for either name, any name will match.) For security reasons, **.rhosts** must belong to the user granting the equivalence or to root.

The file **/etc/hosts.equiv** is a list of remote hosts with matching-name equivalence. The file lists remote hosts one per line. On each host listed in **/etc/hosts.equiv**, a remote user with the same name as a local user is equivalent to the local user. In effect, the users are the same if the names are the same.

FILES

$HOME/.rhosts
/etc/hosts.equiv

SEE ALSO

rcmd(1), rcp(1), rlogin(1).
CTIX Network Administrator's Guide.

WARNINGS

When a system is listed in **/etc/hosts.equiv**, its security must be as good as local security. One insecure system mentioned in **/etc/hosts.equiv** can compromise the security of an entire network.

NAME

rmtab - remotely mounted file system table

DESCRIPTION

Rmtab resides in directory */etc* on the server and contains a record of all clients that have done remote mounts of file systems from this server. Whenever a remote *mount* is done from the client, an entry is made in the *rmtab* file of the host serving up that file system. *Umount* from the client removes NFS remote mount entries from the table. If the client crashes, all entries for client will be removed by *showmount -r* when the client reboots. The table is a series of lines of the form

hostname:directory

This table is used only to preserve information between crashes, and is read only by *mountd*(1M) when it starts up. *Mountd* keeps an in-core table, which it uses to handle requests from programs like *showmount*(1).

FILES

/etc/rmtab

SEE ALSO

showmount(1), mountd(1M), mount(1M), umount(1M).

BUGS

Although the *rmtab* table is close to the truth, it is not 100% accurate.

NAME

rpc - Sun rpc program number database

SYNOPSIS

/etc/rpc

DESCRIPTION

The *rpc* file contains user readable names of rpc (Remote Procedure Call) services that can be used in place of rpc program numbers. It is used by programs such as *rpcinfo*(1M).

Each line is of the following format:

rpc_program_server_name rpc_program_number aliases...

Items are separated by any number of blanks and/or tab characters. A # character indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

FILES

/etc/rpc

SEE ALSO

getrpcent(3).
CTIX Network Programmer's Primer.

NAME

rtab - Remote I/O Processor configuration table

DESCRIPTION

The RIOP table file, **rtab**, defines which RIOPs are known to the system. This file resides in the directory **/etc/riop.** Each entry in this table consists of one line with three ASCII fields separated by a colon (:) and gives information about one RIOP in the system.

The first field is the unique identification number which is coded into an ID prom on the RIOP. In the RIOP, this is a 4 byte number; the most significant byte is the product code, which is always 0x20 for an RIOP. The *rtab* field should only contain the lower 3 significant bytes of this number and be expressed in hexadecimal.

The second field is the decimal ordinal number for the RIOP. This field is used to order each RIOP from 0 to 31 such that RIOP #0 is related to the first group of 16 virtual ttys, RIOP #1 is related to the second group of 16 virtual ttys, and so on. The numbers of this field in different lines of the file do not have to be sequentially ordered, although this is recommended for ease of administration.

The third field is the version number suffix string which is appended to the string "**/etc/riop/riop**" by the RIOP daemon to form the full path name of the executable object file to be downloaded into the RIOP. For the first release, this field contains "**1.00**". This mechanism allows for the simultaneous use of multiple RIOPs operating at different download image release levels.

An optional fourth comment field may be added to each line by appending a colon (:) immediately after the version string, followed by text up to the newline.

FILES

/etc/riop/rtab

SEE ALSO

riopcfg(1M), riopqry(1M).

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 which 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:

@hDDDDD

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.

Delta table

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

@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 (currently, normal: **D**, and 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 newlines. 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 will cause 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 [*get*(1) with the **-e** keyletter]. The **q** flag defines the replacement for the **%Q%** identification keyword. The **z** flag is used in certain 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 the following in respective order:

@I DDDDD
@D DDDDD
@E DDDDD

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

scnhdr - section header for a common object file

SYNOPSIS

#include <scnhdr.h>

DESCRIPTION

Every common object file has a table of section headers to specify the layout of the data within the file. Each section within an object file has its own header. The C structure appears below.

```
struct      scnhdr
{
        char            s_name[8]; /* section name */
        long            s_paddr;   /* physical address */
        long            s_vaddr;   /* virtual address */
        long            s_size;    /* section size */
        long            s_scnptr;  /* file ptr to raw data */
        long            s_relptr;  /* file ptr to relocation */
        long            s_lnnoptr; /* file ptr to line numbers */
        unsigned short  s_nreloc;  /* # reloc entries */
        unsigned short  s_nlnno;   /* # line number entries */
        long            s_flags;   /* flags */
};
```

File pointers are byte offsets into the file; they can be used as the offset in a call to FSEEK [see *ldfcn*(4)]. If a section is initialized, the file contains the actual bytes. An uninitialized section is somewhat different. It has a size, symbols defined in it, and symbols that refer to it. But it can have no relocation entries, line numbers, or data. Consequently, an uninitialized section has no raw data in the object file, and the values for *s_scnptr*, *s_relptr*, *s_lnnoptr*, *s_nreloc*, and *s_nlnno* are zero.

SEE ALSO

ld(1), fseek(3S), a.out(4).

NAME

scr_dump - format of curses screen image file.

SYNOPSIS

scr_dump(file)

DESCRIPTION

The *curses*(3X) function *scr_dump*() will copy the contents of the screen into a file. The format of the screen image is as described below.

The name of the tty is 20 characters long and the modification time (the *mtime* of the tty that this is an image of) is of the type *time_t*. All other numbers and characters are stored as *chtype* (see **<curses.h>**). No newlines are stored between fields.

<magic number: octal 0433>
<name of tty>
<mod time of tty>
<columns> <lines>
<line length> <chars in line> for each line on the screen
<line length> <chars in line>
.
.
.
<labels?> 1, if soft screen labels are present
<cursor row> <cursor column>

Only as many characters as are in a line will be listed. For example, if the *<line length>* is **0**, there will be no characters following *<line length>*. If *<labels?>* is TRUE, following it will be

<number of labels>
<label width>
<chars in label 1>
<chars in label 2>
.
.
.

SEE ALSO

curses(3X).

NAME

services - list of Internet services

DESCRIPTION

The file **/etc/services** lists known DARPA Internet services. Each line describes a single service and consists of the following blank separated fields:

name number/protocol aliases ...

where

name	is the official name of the service.
number	is the service number.
protocol	is the name of the protocol (see *protocols*(4)) used by the service.
aliases ...	is a blank-separated list of local aliases for the service.

The routines that search this file ignore comments (portions of lines beginning with #) and blank lines.

Service names and numbers are specified by the DDN Network Information Center. Do not change this file unless you are familiar with DARPA Internet internals.

FILES

/etc/services.

SEE ALSO

CTIX Network Administrator's Guide.

NAME

shadow - shadow password file

DESCRIPTION

The *shadow* file contains the following information for each user:

- login name
- encrypted password
- aging information

The aging information includes three integer fields:

lastchange The number of days from the epoch (midnight, 1/1/70) to the last time the password was changed.

mindays The minimum number of days between password changes, defined as MINWEEKS in **/etc/default/passwd**.

maxdays The number of days the password is valid, defined as MAXWEEKS in **/etc/default/passwd**.

If *mindays* and *maxdays* equal 0, the user must change the password at the next login. If *mindays* is greater than *maxdays*, only the super-user can change the password.

This is an ASCII file. Each field within each user's entry is separated from the next by a colon. The file resides in the **/etc** directory and can be read only by the super-user.

FILES

/etc/passwd
/etc/opasswd
/etc/oshadow

SEE ALSO

login(1), passwd(1), pwconv(1), getpwent(3X) getspent(3X), passwd(4).

NAME

syms - common object file symbol table format

SYNOPSIS

#include <syms.h>

DESCRIPTION

Common object files contain information to support symbolic software testing [see *sdb*(1)]. Line number entries, *linenum*(4), and extensive symbolic information permit testing at the C *source* level. Every object file's symbol table is organized as shown below.

```
File name 1.
        Function 1.
                Local symbols for function 1.
        Function 2.
                Local symbols for function 2.
        ...
        Static externs for file 1.

File name 2.
        Function 1.
                Local symbols for function 1.
        Function 2.
                Local symbols for function 2.
        ...
        Static externs for file 2.

...

Defined global symbols.
Undefined global symbols.
```

The entry for a symbol is a fixed-length structure. The members of the structure hold the name (null padded), its value, and other information. The C structure is given below.

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

struct syment
{
  union                          /* all ways to get symbol name */
  {
```

```
        char            _n_name[SYMNMLEN];
                                    /* old COFF version */
        struct
        {
          long          _n_zeroes;  /*new == 0 */
          long          _n_offset;  /* offset into string table */
        } _n_n;
        char            *_n_nptr[2]; /* allows overlaying */
    } _n;
    long                n_value;    /* value of symbol */
    short               n_scnum;    /* section number */
    unsigned short      n_type;     /* type and derived type */
    char                n_sclass;   /* storage class */
    char                n_numaux;   /* number of aux entries */
};

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

Meaningful values and explanations for them are given in both **syms.h** and *Common Object File Format*. Anyone who needs to interpret the entries should seek more information in these sources. Some symbols require more information than a single entry; they are followed by *auxiliary entries* that are the same size as a symbol entry. The format follows.

```
union auxent
{
        struct
        {
                long            x_tagndx;
                union
                {
                        struct
                        {
                                unsigned short x_lnno;
                                unsigned short x_size;
                        } x_lnsz;
                        long    x_fsize;
                } x_misc;
                union
                {
```

```
                        struct
                        {
                                long     x_lnnoptr;
                                long     x_endndx;
                        }       x_fcn;
                        struct
                        {
                                unsigned short x_dimen[DIMNUM];
                        }       x_ary;
                }               x_fcnary;
                unsigned short x_tvndx;
        }       x_sym;
        struct
        {
                char    x_fname[FILNMLEN];
        }       x_file;
        struct
        {
            long    x_scnlen;
            unsigned short x_nreloc;
            unsigned short x_nlinno;
        }   x_scn;

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

Indexes of symbol table entries begin at *zero*.

SEE ALSO

sdb(1), a.out(4), linenum(4).
UNIX System V Release 3.2 Programmer's Guide.

WARNINGS

CTIX C longs are equivalent to ints and are converted to ints in the compiler to minimize the complexity of the compiler code generator. Thus the information about which symbols are declared as longs and which, as ints, does not show up in the symbol table.

NAME

system - system description file

DESCRIPTION

The system description describes tunable variables and hardware configuration of the CTIX system.

The file is formatted in sections. Each section begins with a section header (an ! followed by a single word). Each section varies in format, depending on the format required by the program that uses the data provided by that section.

Note that with respect to the **!TUNEABLES** section, changes made to this section do not take effect until the *uconf*(1M) program is run.

In the example file below, the **!TUNEABLES** section describes a cluster terminal configuration where only two cluster lines are used and there are six ttys associated with each line: Cluster line 0 has tty256-261 and Cluster line 1 has tty262-267. (Note that *uconf* must be run in order for this configuration to take effect.)

The **!VMESLOTS** section of the same example file describes the VME boards for the EEPROM. The *slot* field is the slot position in the VME bus. The *type* field is the board type, any of the following:

1 CMC Ethernet board

2 Interphase SMD disk controller board

4 Interphase half-inch tape controller board

5 Multiprotocol Communications Controller board

The *address* field is the location of the board. The *length* field is the address space size of the board. The optional *initialization function name* is an initialization function called by the PROM at boot time.

The **!VMECODE** section consists of a list of files that describe the executable code to be loaded into the EEPROM. This section is required only if a bootable initialization function is specified.

The **!SCSIMAP** section consists of several one-line (up to 64 characters) entries, each specifying a logical-to-physical mapping for a SCSI device. The disk controller number must always be **c0**.

The range of drive numbers in the !SCSIMAP follows:

Drive Type	Drive Number Range
disk	d0 .. d9 and da .. df
tape	d0 .. d7

The range of target numbers for each bus is **0** through **6**. Target number **7** is reserved for the host ID.

The range of bus numbers for each type of system follows:

System Type	Bus	SCSI Controller
S/80, S/280	0	onboard SCSI
S/120, S/22*x*, S/320	1 .. 4	four SCSI RS-232 boards
S/480, S/640	0 .. 4	onboard SCSI and four SCSI RS-232 boards

EXAMPLE

```
!FILENAMES
PROM_IFILE=/etc/lddrv/EEPROM.ifile
EEPROM_FILE=/dev/vme/eeprom
!TUNEABLES
cl_deflines=2
cl_defdrops=6
!VMESLOTS
* The following section describes the VME boards
*
*slot     type  address     length   [initialization
*                                      function name ]
*
0         2     C1000000    512      loadvs32
1         2     C1000200    512
*one CMC Ethernet controller)
2         1     C0DE0000    131072
*
!VMECODE
/etc/lddrv/DISKVS32.o
!SCSIMAP
disk-c0d0 bus=0    target=6  lun=0  parity  reselect
disk-c0d1 bus=0    target=5  lun=0  parity  reselect
tape-d0   bus=0    target=1  lun=0  parity  reselect
tape-d1   bus=0    target=2  lun=0  parity  reselect halfinch
disk-c0d2 bus=1    target=6  lun=0  parity  reselect
```

```
disk-c0d3 bus=1    target=5   lun=0   parity   reselect
disk-c0d4 bus=1    target=4   lun=0   parity   reselect
tape-d2   bus=2    target=0   lun=0   parity   reselect halfinch
```

FILES

/etc/system

SEE ALSO

lddrv(1M), ldeeprom(1M), scsimap(1M), uconf(1M), vme(7).
S/Series CTIX Administrator's Guide.

NOTE

On an S/80, S/280, or S/480: on bus 0, disk-c0d0 is the **rootdev**. On an S/80, on bus 0, target 0 is reserved for the Ethernet LANCE chip [see *scsimap*(1M)]. In both cases, the !SCSIMAP entries *should not be changed*.

NAME

tapedrives - tape drive specific information used by the /etc/tapeset command.

DESCRIPTION

The **/etc/tapedrives** file contains tape drive- and controller-specific information that the *tapeset*(1M) command uses to configure drives with controllers.

Each entry in the **/etc/tapedrives** file is a line of the following form:

drive_name *ctrl_type* *max_blocksize* *ctrl_flags*

where

drive_name Corresponds to the drive name used in the **-t** option of *tapeset*(1M).

ctrl_type Specifies the controller type: **i** for the Interphase V/Tape controller or **s** for a SCSI controller.

max_blocksize Specifies the decimal value of the maximum size block the drive can accept.

ctrl_flags Are controller-specific flags, the format of which depends on the value of *ctrl_type*.

VME Controller-Based Drive Flags

Flags for the Interphase V/Tape controller follow:

density_flags *speed_flags* *gap_flags*

Any of the *density_flags*, *speed_flags*, or *gap_flags* can be omitted, and the flags can be specified in any order.

The **/usr/include/sys/iptioctl.h** header file describes each flag.

The format for the density, speed, and gap flags follows:

flag [| *flag* ...][, *nn*...]

where

flag Can be any of the following:
DSBOK, DSB, DSBL, DSBFLGS, SSBOK, SSB, SPD, SPDL, SPDFLGS, GSBOK, GSB, LGAP, LGAPL, GAPFLGS

nn Can be two or three hexadecimal numbers that correspond to the formatter commands used to change density (high, med, low), speed (high, low), or gap (default, extended) if the other flags specify that the controller expects formatter commands.

SCSI Controller-Based Drive Flags

In the case of SCSI controller-based tape drives, the flags are used to set drive parameters. The SCSI drive controller is first interrogated for the existing mode-sense data. The flags follow:

data_length *mode_sense_data mode_select_data*

where

data_length	Is the size of the mode-sense data block in bytes
mode_sense_data	Is the mask that is ORed with the mode-sense data block read from the drive
mode_select_data	Is the mask that is ANDed with the result of the ORed *mode_sense_data mask* and the mode-sense data block; the two-step result is used to reconfigure the drive.

If *data_length* is a non-zero decimal value, then *mode_sense_data* and *mode_select_data* must contain the correct number of hexadecimal digits with no spaces.

EXAMPLES

```
* Dumb half-inch tape drive
dumb      i   131072
*
dumb-64   i   65536
*
* Cipher M990 GCR CacheTape Drive
* Manual Reference:  M990 GCR CacheTape Unit
* Maintenance Manual Fourth Edition
* Note: Must have VME Eprom version 004.
*            Interphase tape controller
*            64K max block size
*            Sets density via formatter command
*                (low=16,med=17,high=09)
*            Speed can be set via J1-36
*            M990 is low speed, M990-hs is high speed
*
M990      i   65536   DSBOK | DSB,16,17,09      SSBOK
M990-hs   i   65536   DSBOK | DSB,16,17,09      SSBOK | SPD
*
* Cipher F880 Microstreamer Tape Drive.
* Manual Reference: F880 Series Microstreamer Tape
* Drive Product Description
```

```
*           Interphase tape controller
*           64K max block size
*           Sets speed via J2-50
*           F880 is low speed, F880-hs is high speed
*
F880        i    131072   SSBOK | SPDL
F880-hs     i    131072   SSBOK | SPDL | SPD
*
*
*Dumb SCSI tape drive
*
*           (Since all SCSI drives can run with blocksize of 128K -
*           they break it up into many 512 byte blocks - the field
*           is set to 128K, but it will be ignored by the tapeset
*           command.)
*
dumb        s    131072
*
*
*Archive 5945S (archive + emulex controller) SCSI tape drive
*           Use mode select to turn off auto load (bit 2, byte 13).
*Cipher F880S and M990S (F880 and M990 with SCSI Adapter)
*
5945S-noauto s 131072 13 00000000000000000000000002 ffffffffffffffffffffffffff
5945S-auto s 131072 13 00000000000000000000000000 fffffffffffffffffffffffffd
F880S s 131072 12 000000000000000000000000 00007fffffffffff00000000
M990S s 131072 12 000000000000000000000000 00007fffffffffff00000000
*
```

SEE ALSO

tapeset(1M), ipt(7), qic(7).

NAME

term - format of compiled term file.

SYNOPSIS

/usr/lib/terminfo/?/*

DESCRIPTION

Compiled *terminfo*(4) descriptions are placed under the directory */usr/lib/terminfo*. In order to avoid a linear search of a huge CTIX system directory, a two-level scheme is used: */usr/lib/terminfo/c/name* where *name* is the name of the terminal, and *c* is the first character of *name*. Thus, **act4** can be found in the file */usr/lib/terminfo/a/act4*. Synonyms for the same terminal are implemented by multiple links to the same compiled file.

The format has been chosen so that it will be the same on all hardware. An 8-bit byte is assumed, but no assumptions about byte ordering or sign extension are made. Thus, these binary *terminfo*(4) files can be transported to other hardware with 8-bit bytes.

Short integers are stored in two 8-bit bytes. The first byte contains the least significant 8 bits of the value, and the second byte contains the most significant 8 bits. (Thus, the value represented is 256*second+first.) The value **-1** is represented by **0377,0377**, and the value **-2** is represented by **0376,0377**; other negative values are illegal. The **-1** generally means that a capability is missing from this terminal. The **-2** means that the capability has been cancelled in the *terminfo*(4) 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*(1M)] by using the *terminfo*(4) compiler, *tic*(1M), and read by the routine **setupterm**( ). [See *curses*(3X).] The file is divided into six parts: the header, terminal names, boolean flags, numbers, strings, and string table.

The header section begins the file. This section contains six short integers in the format described below. These integers are (1) the magic number (octal **0432**); (2) the size, in bytes, of the names section; (3) the number of bytes in the boolean section; (4) the number of short integers in the numbers section; (5) the number of offsets (short integers) in the strings section; (6) the size, in bytes, of the string table.

The terminal names section comes next. It contains the first line of the *terminfo*(4) description, listing the various names for the terminal, separated by the bar (|) character [see *term*(5)]. The section is terminated with an ASCII NUL character.

The boolean flags have one byte for each flag. This byte is either **0** or **1** as the flag is present or absent. The value of **2** means that the flag has been cancelled. The capabilities are in the same order as the file **<term.h>**.

Between the boolean section and the number section, a null byte will be inserted, if necessary, to ensure that the number section begins on an even byte. All short integers are aligned on a short word boundary.

The numbers section is similar to the 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, an octal dump of the description for the Microterm ACT 4 is included:

```
microterm|act4|microterm act iv,
   cr=^M, cud1=^J, ind=^J, bel=^G, am, cub1=^H,
   ed=^_, el=^^, clear=^L, cup=^T%p1%c%p2%c,
   cols#80, lines#24, cuf1=^X, cuu1=^Z, home=^],
```

```
000 032 001   \0 025 \0 \b \0 212 \0 " \0 m i c r
020 o t e r m | a c t 4 | m i c r o
040 t e r m   a c t   i v \0 \0 001 \0 \0
060 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0 \0
100 \0 \0 P \0 377 377 030 \0 377 377 377 377 377 377 377 377
120 377 377 377 377 \0 \0 002 \0 377 377 377 377 004 \0 006 \0
140 \b \0 377 377 377 377 \n \0 026 \0 030 \0 377 377 032 \0
160 377 377 377 377 034 \0 377 377 036 \0 377 377 377 377 377 377
200 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
520 377 377 377 377   \0 377 377 377 377 377 377 377 377 377 377
540 377 377 377 377 377 377 007 \0 \r \0 \f \0 036 \0 037 \0
560 024 % p 1 % c % p 2 % c \0 \n \0 035 \0
600 \b \0 030 \0 032 \0 \n \0
```

Some limitations: total compiled entries cannot exceed 4096 bytes; all entries in the name field cannot exceed 128 bytes.

FILES

/usr/lib/terminfo/?/*	compiled terminal description database
/usr/include/term.h	*terminfo*(4) header file

SEE ALSO

infocmp(1M), curses(3X), terminfo(4), term(5).
UNIX System V Release 3.2 Programmer's Guide.

NAME

termcap - terminal capability database

SYNOPSIS

/etc/termcap

DESCRIPTION

This entry describes terminal-independent programming conventions that originate at UC Berkeley. UNIX System V initially borrowed *termcap* but has since changed to the *terminfo*(4) convention. CTIX continues to support *termcap* so as to be compatible with the Berkeley version of the UNIX system. But use *terminfo* in new programs.

termcap programs work from information supplied through the **TERM** and **TERMCAP** environment variables. The location of the description depends on the value of **TERMCAP**:

- If **TERMCAP** is not set or is empty, **TERM** is the name of a description in */etc/termcap*.
- If **TERMCAP** has a value that begins with a /, **TERM** is the name of a description in the file named by **TERMCAP**.
- If **TERMCAP** begins with any character except /, **TERMCAP** contains the description.

A description begins with a list of its names separated by vertical bars. The rest of the description is a list of capabilities separated by colons. If you use more than one line, precede each newline except the last with :\ Here's a simple example.

```
d5|vt50|dec vt50:\
        :bs:cd=\EJ:ce=\EK:cl=\EH\EJ:co#80:li#12:\
        :nd=\EC:pt:up=\EA:
```

There are three kinds of capabilities:

- *Boolean*. These indicate the presence or absence of a terminal feature by their presence or absence. Boolean capabilities consist of two characters (the capability name).
- *Numeric*. These indicate some numeric value for the terminal, such as screen size or delay required by a standard character. Numeric capabilities consist of two characters (the capability name), followed by a #, followed by a decimal number.

- *String*. These indicate a sequence that performs some operation on the terminal. String capabilities consist of two characters (the capability name), optionally followed by a delay, followed by a string.

  The delay is the number of milliseconds the program must wait after using the sequence; specify no more than one decimal place. If the delay is proportional to the number of lines affected, end it with a *.

  The string is a sequence of characters. The following sequences are specially interpreted.

\E	Escape Character
^*x*	Control-*x*
\n	Newline
\r	Return
\t	Tab
\b	Backspace
\f	Formfeed
xxx	Octal value of *xxx*
\072	: in string
\200	null (\000 doesn't work)

  Octal numbers must be three digits long.

  Some strings are interpreted further, such as **cm**.

You can follow any capability name with an @, to indicate that the terminal lacks the capability. This is only useful in conjunction with the **tc** capability; see "Similar Terminals," below.

Here is a list of standard capabilities. (P) indicates a string that might require padding; (P*) indicates a string that might require proportional padding.

Name	Type	Pad?	Description
ae	str	(P)	Ends alternate character set.
al	str	(P*)	Adds new blank line.
am	bool		Terminal has automatic margins.
as	str	(P)	Starts alternate character set.
bc	str		Backspace if not Control-h.
bs	bool		Terminal can backspace with Control-h.
bt	str	(P)	Back tab.
bw	bool		Backspace wraps from column 0 to last column.
CC	str		Command character in prototype if terminal settable.
cd	str	(P*)	Clears to end of display.

ce	str	(P)	Clears to end of line.
ch	str	(P)	Moves cursor horizontally to specified column.
cl	str	(P*)	Clears screen.
cm	str	(P)	Moves cursor to specified row and column.
co	num		Number of columns in a line.
cr	str	(P*)	Carriage return if not Control-m.
cs	str	(P)	Change scrolling region.
cv	str	(P)	Moves cursor vertically to specified row.
da	bool		Display can be retained above.
dB	num		Delay after backspace, in milliseconds.
db	bool		Display can be retained below.
dC	num		Delay after carriage return, in milliseconds.
dc	str	(P*)	Delete character.
dF	num		Delay after form feed, in milliseconds.
dl	str	(P*)	Deletes line.
dm	str		Enters delete mode.
dN	num		Delay after newline, in milliseconds.
do	str		Goes down one line.
dT	num		Delay after tab, in milliseconds.
ed	str		Ends delete mode.
ei	str		Ends insert mode; give an empty string if you've defined **ic**.
eo	str		Can erase overstrikes with a blank.
ff	str	(P*)	Hardcopy terminal page eject if not form feed.
hc	bool		Hardcopy terminal.
hd	str		Half-line down (forward 1/2 linefeed).
ho	str		Move cursor to upper left corner (home).
hu	str		Half-line up (reverse 1/2 linefeed).
hz	str		Hazeltine or other terminal that can't print ~'s.
ic	str	(P)	Insert character.
if	str		Name of file containing terminal initialization.
im	bool		Starts insert mode; give an empty string if you've defined **ic**.
in	bool		Insert mode distinguishes nulls on display.
ip	str	(P*)	Pad after insertion.
is	str		Terminal initialization.
k0-k9	str		Sent by special (usually numeric) function keys. If programmable, set with **is**, **if**, **vs**, or **ti**.
kb	str		Sent by backspace key.
kd	str		Sent by terminal down arrow key.

ke	str		Ends keypad transmit mode.
kh	str		Sent by home key.
kl	str		Sent by terminal left arrow key.
kn	num		Number of special function keys.
ko	str		Terminal capabilities that have keys.
kr	str		Sent by terminal right arrow key.
ks	str		Begin keypad transmit mode.
ku	str		Sent by terminal up arrow key.
l0-l9	str		Labels on special function keys.
li	num		Number of lines on screen or page.
ll	str		Last line, first column.
ma	str		Command key map; used by ex version 2 (Convergent uses version 3).
mi	bool		Safe to move while in insert mode.
ml	str		Memory lock on above cursor.
ms	bool		Safe to move while in standout and underline mode.
mu	str		Memory unlock (turn off memory lock).
nc	bool		No correctly working carriage return (DM2500,H2000).
nd	str		Non-destructive space (cursor right).
nl	str	(P*)	Begin a new line if not newline.
ns	bool		A video terminal that doesn't scroll!
os	bool		Terminal overstrikes.
pc	str		Pad character if not null.
pt	bool		Has hardware tabs; if they need to be set put sequence in is or if.
se	str		Ends stand out mode.
sf	str	(P)	Scrolls forwards.
sg	num		Number of blank chars left by so or se.
so	str		Begins stand out mode.
sr	str	(P)	Scroll reverse (backwards).
ta	str	(P)	Tab if not Control-i or with padding.
tc	str		Name of terminal that has some of the same capabilities; tc must be the last capability.
te	str		Ends programs that do cursor motion.
ti	str		Initializes programs that do cursor motion.
uc	str		Underscores and moves past one character.
ue	str		Ends underscore mode.
ug	num		Number of blank spaces that surround underscore mode.
ul	bool		Terminal underlines automatically even though it can't overstrike

up	str	Upline (cursor up).
us	str	Start underscore mode.
vb	str	Visible bell (must not move cursor).
ve	str	Ends open and visual modes.
vs	str	Initializes open and visual modes.
xb	bool	Beehive (f1=escape, f2=ctrl C).
xn	bool	Terminal ignores newline after wrap (Concept).
xr	bool	Return clears to end of line and goes to beginning of next line (Delta Data).
xs	bool	Writing on standout mode text produces standout mode text (HP 264?).
xt	bool	Destructive tabs, magic standout character (Teleray 1061).

Pointers on Preparing Descriptions

- You may want to copy the description of a similar terminal.
- Build up a description gradually, checking partial descriptions with *ex*.
- Be aware that an unusual terminal may expose bugs in *ex*. limitations in the *termcap* convention.

Basic Capabilities

The following capabilities are common to most terminals. The **co** capability gives the number of columns per line. The **li** gives the number of lines on a video terminal. The **am** capability indicates that writing off the right edge takes the cursor to the beginning of the next screen. The **cl** capability tells how the terminal clears its screen. The **bs** indicates that the terminal can backspace; but if the terminal doesn't use Control-h, specify **bc** instead of **bs**. The **os** capability indicates that printing a character at an occupied position doesn't destroy the existing character.

A couple of notes on moving off the edge. Programs that use this convention never move the cursor off the top or the left edge of the screen. On the other hand, they assume that moving off the bottom edge scrolls the display up.

These capabilities suffice to describe hardcopy and very dumb terminals. For example, the Teletype Model 33 has this description.

```
t3|33|tty33:co#72:os
```

This is LSI ADM3 (without the cursor addressing option).

```
cl|adm3|3|lsi adm3:am:bs:cl=^Z:li#24:co#80
```

Cursor Addresses and Other Variables

If a string capability includes a variable value, use a **%** escape to indicate the value. By default, programs take these values to be zero origin (that is, the first possible value is 0) and that the **cm** capability specifies two values: row, then column. Use the **%r** or **%i** capability if either assumption is incorrect.

These are the valid **%** escapes.

%d	print the values as a decimal number
%2	print the values as a two-digit decimal number
%3	print the values as a three-digit decimal number
%.	print the value in binary (but see below)
%+*x*	add ASCII value of *x* to value, then print in binary
%>*xy*	if the next value is greater than the ASCII value of *x*, add the ASCII value of *y* before using the value's **%** escape
%r	row is the first value in this **cm**
%i	values are 1-origin
%%	print a %
%n	in this capability, exclusive or the values with 01400 before using the values' **%** escapes (DM2500)
%B	change the next value to binary coded decimal ((16*(*x*/10)) + (*x*%10) where *x* is the value) before interpreting it
%D	The next value is reverse-coded (*x*-2*(*x*%16) where *x* is the value; Delta Data)

A program should avoid using a **cm** sequence that includes a tab, newline, Control-d, or carriage return, because the terminal interface may misinterpret these characters. If possible, use the **cm** sequence to move to the row or column after the destination, then use local motion to get to the destination.

Here are some examples of **cm** definitions. To position the cursor of an HP2645 on row 3, column 12, you must send the terminal ''\E&a12c03Y'', followed by a 6 millisecond delay; the HP2645 description includes **:cm=6\E&%r%2c%2Y:**. To position the cursor of an ACT-IV, you send it a Control-t, followed by the row and column in binary; the ACT-IV description includes **:cm=^T%.%.:** The LSI ADM3a uses the set of printable ASCII characters to represent row and column values; its description includes **:cm\E=%+%+:**.

Local and General Cursor Motions

Most terminals have short strings that trigger commonly-used cursor motions. A non-destructive space (BR nd) moves the cursor one position right. An upline sequence (**up**) moves the cursor one position up. A home sequence (**ho**) moves the cursor to the upper left hand corner. A lower-left (**ll**) goes to the other lefthand corner. The **ll** capability may be a sequence that moves the cursor home, then up; but otherwise programs never do this.

Area Clears

Some terminals have short sequences that clear all or part of a display. Clear (**cl**) clears the screen and homes the cursor; if clearing the screen does not restore the terminal's normal modes, **cl** should include the strings that do. Clear to end of line (**ce**) clears from the current cursor position to the right. Clear to end of display (**cd**) clears from the current cursor position to the bottom of the display; programs always move the cursor to the beginning of the line before using **cd**.

Insert/Delete Line

Many terminals have strings that shift text starting at the current cursor position. Programs always move the cursor to the beginning of the line before using these strings. Add line (**al**) shifts the current line and all below it down a position leaving the cursor on the newly-blanked line. Delete line (deletes the line the cursor is on without moving the cursor. If a terminal description has an **al** capability, you do not really need to specify **sb**.

If deleting a line might produce a non-blank line at the bottom of the screen, specify **db**. If scrolling backwards might produce a non-blank line at the top of the screen, specify **da**.

Insert/Delete Character

The termcap convention recognizes two kinds of terminal insert/delete string.

- The first convention is by far more common. Using insert or delete modes only affect characters on the current line. Inserting a single character shifts all characters, including all blanks, to the right; the character on the right edge of the screen is lost. No special capability is required to describe this kind of terminal.
- The second convention is more rare and complicated. The terminal distinguishes between blank spaces created by output tabs (011) or spaces (040) from all other blanks; other blanks are known as nulls. Inserting a character eliminates the first null to the right of the cursor; deleting a character doubles the first null. If there are no nulls on the current line inserting a character inserts the line's rightmost character at the beginning of the next line. Use the **in** capability to describe this kind of terminal.

Notable among the second type are the Concept 100 and the Perkin Elmer Owl.

A simple experiment shows what type you have. Set the terminal to its "local" mode. Clear the screen, then type a short sequence of text. Move the cursor to the right several spaces *without using the space or tab characters.* Type a second short sequence of text. Move the cursor back to the beginning of the first text. Start the terminal's insert mode and begin tapping the Spacebar. If you have the first kind of terminal, both sequences of text will move at once, at whatever character is at the right edge of the screen will be lost. If you have the second kind of terminal, at first only the first sequence of text will move; when the first sequence hits the second sequence, it will push the second onto the next line.

A terminal can have either an insert mode or the ability to insert a single character. Specify insert mode with **im** and **ei**. To specify that the terminal can insert a single character, specify **ic** *and* specify empty strings for **im** and **ei**. If you must delay or output more control text after inserting a single character, specify **ip**.

If a terminal has both an insert mode and the ability to insert a single character, it is usually best not to specify **ic**.

Some programs operate more quickly if they are allowed to move the cursor around randomly while in insert mode. For example, *vi* has to delete a character when you insert a character before a tab. If your terminal permits this, specify move on insert **mi**. Beware of terminals that foul up in subtle ways when you do this (the Datamedia, for example).

Delete mode (**dm**), end delete mode (**ed**), and delete character (**dc**) work like **im**, **ei**, and **ic**.

Highlighting, Underlining, and Visible Bells

Specify the terminals most distinctive display mode with **so se**. Half intensity is usually not a good choice unless the terminal is normally in reverse video.

The convention provides for underline mode and for single character underlining. Specify underline mode with **us** and **ue**. Specify a way to underline and move past a character with **uc**; if your terminal can underline a single character but doesn't automatically move on, add a nondestructive space to the **uc** string.

Some terminals can't overstrike but still correctly underline text without special help from the host computer. If yours is one, specify **ul**.

If your terminal spaces before and after entering standout and underline mode, specify **ug**.

Programs leave standout and underline mode before moving the cursor or printing a newline.

If the terminal can flash the screen without moving the cursor, specify **vb** (visual bell).

If the terminal needs to change working modes before entering the open and visual modes of *ex* and *vi*, specify **vs** and **ve.** respectively. These can be used to change, for example, from a underline to a block cursor and back.

If the terminal needs to be in a special mode when running a program that addresses the cursor, specify **ti** and **te**. This may be important if a terminal has more than one page of memory. If the terminal has memory-relative cursor addressing but not screen relative cursor addressing, use **ti** to fix a screen-sized window into the terminal.

If a terminal can overstrike, programs assume that printable spaces don't destroy anything, unless you specify **eo**.

Keypad

Some terminals have keypads that transmit special codes. If the keypad can be turned on and off, specify **ks** and **ke**; if you don't, programs assume that the keypad is always on. Specify the codes sent by cursor motion keys with **kl**, **kr**, **ku**, **kd**, and **kh**. If there are function keys specify the codes they send with **f1**, **f2**, **f3**, **f4**, **f5**, **f6**, **f7**, **f8**, and **f9**. If these keys have labels other than the usual ''f0 through'' ''f9'', specify the labels **l1**, **l2**, **l3**, **l4**, **l5**, **l6**, **l7**, **l8**, and **l9**. If there are other keys that transmit the same code that the terminal expects for a function, such as clear screen, mention the affected capabilities in the **ko** capability. For example, '':ko=cl,ll,sf,sb:'' says that the terminal has clear, home down, scroll down, and scroll up keys that transmit the same thing as the cl, ll, sf, and sb capabilities.

Terminal Initialization

If a terminal must be initialized, on login for example, specify a short string with **is** or a file containing initialization strings with **if**. Other capabilities include **is**, an initialization string for the terminal, and **if**, the name of a file containing long initialization strings. If both are given, **is** is printed before **if**. If the terminal has tab stops, these strings should first clear all stops, then set new stops at the 9 column and every 8 columns thereafter.

Similar Terminals

If a new terminal strongly resembles an existing terminal, you can write a description of the new terminal that only mentions the old terminal and the capabilities that differ. The **tc** capability describes the old terminal; it must be the last capability in the description. If the old terminal has capabilities that the new one lacks, specify an @ *after the capability name.*

The different entries you create with **tc** need not represent terminals that are actually different. They can represent different uses for a single terminal, or user preferences as to which terminal features are desirable.

The following example defines a describes a variant of the *2621* that never turns on the keypad.

```
hn|2621nl:ks@:ke@:tc=2621:
```

FILES

/etc/termcap standard database

SEE ALSO

ex(1), more(1), tset(1), ul(1), vi(1), ocurse(3X), otermcap(3X), terminfo(4).

BUGS

ex allows only 256 characters for string capabilities, and the routines in *otermcap*(3X) do not check for overflow of this buffer.

The total length of a single description (excluding only escaped newlines) may not exceed 1024 characters. If you use **tc**, the combined description may not exceed 1024 characters.

The **vs**, and **ve** entries are specific to the *vi* program.

Not all programs support all entries. There are entries that are not supported by any program.

The **ma** capability is obsolete and serves no function in our database; Berkeley includes it for the benefit of systems that cannot run version 3 of *vi*.

NAME

terminfo - terminal capability database

SYNOPSIS

/usr/lib/terminfo/?/*

DESCRIPTION

terminfo is a compiled database [see *tic*(1M)] describing the capabilities of terminals. Terminals are described in *terminfo* source descriptions by giving a set of capabilities which they have, by describing how operations are performed, by describing padding requirements, and by specifying initialization sequences. This database is used by applications programs, such as *vi*(1) and *curses*(3X), so they can work with a variety of terminals without changes to the programs. To obtain the source description for a terminal, use the **-I** option of *infocmp*(1M).

Entries in *terminfo* source files consist of a number of comma-separated fields. White space after each comma is ignored. The first line of each terminal description in the *terminfo* database gives the name by which *terminfo* knows the terminal, separated by 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 **TERM** 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 can 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**. Modes that the hardware can be in, or user preferences, should be indicated by appending a hyphen and an indicator of the mode. See *term*(5) for examples and more information on choosing names and synonyms.

CAPABILITIES

In the table below, the **Variable** is the name by which the C programmer (at the *terminfo* level) accesses the capability. The **Capname** is the short name for this variable used in the text of the database. It is used by a person updating the database and by the *tput*(1) command when asking what the value of the capability is for a particular terminal. The **Termcap Code** is a two-letter code that corresponds to the old *termcap* capability name.

Capability names have no hard length limit, but an informal limit of 5 characters has been adopted to keep them short. Whenever possible, names are chosen to be the same as or similar to the ANSI X3.64-1979 standard.

Semantics are also intended to match those of the specification.

All string capabilities listed below may have padding specified, with the exception of those used for input. Input capabilities, listed under the **Strings** section in the table below, have names beginning with **key_**. The following indicators may appear at the end of the **Description** for a variable.

(G) indicates that the string is passed through **tparm**( ) with parameters (parms) as given ($\#_i$).

(*) indicates that padding may be based on the number of lines affected.

($\#_i$) indicates the i^{th} parameter.

Variable	Capname	Termcap Code	Description
Booleans:			
auto_left_margin	bw	bw	**cub1** wraps from column 0 to last column
auto_right_margin	am	am	Terminal has automatic margins
no_esc_ctlc	xsb	xb	Beehive (f1=escape, f2=Control-C)
ceol_standout_glitch	xhp	xs	Standout not erased by overwriting (hp)
eat_newline_glitch	xenl	xn	Newline ignored after 80 cols (Concept, vt100)
erase_overstrike	eo	eo	Can erase overstrikes with a blank
generic_type	gn	gn	Generic line type (for example, dialup, switch)
hard_copy	hc	hc	Hardcopy terminal
hard_cursor	chts	HC	Cursor is hard to see
has_meta_key	km	km	Has a meta key (shift, sets parity bit)
has_status_line	hs	hs	Has extra ''status line''
insert_null_glitch	in	in	Insert mode distinguishes nulls
memory_above	da	da	Display may be retained above the screen
memory_below	db	db	Display may be retained below the screen
move_insert_mode	mir	mi	Safe to move while in insert mode
move_standout_mode	msgr	ms	Safe to move in standout modes

needs_xon_xoff	nxon	nx	Padding won't work, xon/xoff required
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
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 tildes(~)
transparent_underline	ul	ul	Underline character overstrikes
xon_xoff	xon	xo	Terminal uses xon/xoff handshaking

Variable	**Capname**	**Termcap Code**	**Description**
Numbers:			
columns	cols	co	Number of columns in a line
init_tabs	it	it	Tabs initially every # spaces
label_height	lh	lh	Number of rows in each label
label_width	lw	lw	Number of cols in each label
line_attribute	ldatt	LA	Line drawing character attribute †
lines	lines	li	Number of lines on screen or page
lines_of_memory	lm	lm	Lines of memory if > **lines**; **0** means varies
magic_cookie_glitch	xmc	sg	Number blank chars left by **smso** or **rmso**
num_labels	nlab	Nl	Number of labels on screen (start at 1)
padding_baud_rate	pb	pb	Lowest baud rate where padding needed
virtual_terminal	vt	vt	Virtual terminal number (not used by CTIX)
width_status_line	wsl	ws	Number of columns in status line

Variable	Capname	Termcap Code	Description
Strings:			
acs_chars	acsc	ac	Graphic charset pairs aAbBcC - default is vt100+
back_tab	cbt	bt	Back tab
bell	bel	bl	Audible signal (bell)
carriage_return	cr	cr	Carriage return (*)
change_scroll_region	csr	cs	Change to lines #1 thru #2 (vt100) (G)
char_padding	rmp	rP	Like **ip** but when in replace mode
clear_all_tabs	tbc	ct	Clear all tab stops
clear_margins	mgc	MC	Clear left and right soft margins
clear_screen	clear	cl	Clear screen and home cursor (*)
clr_bol	el1	cb	Clear to beginning of line, inclusive
clr_eol	el	ce	Clear to end of line
clr_eos	ed	cd	Clear to end of display (*)
column_address	hpa	ch	Horizontal position absolute (G)
command_character	cmdch	CC	Term. settable cmd char in prototype
cursor_address	cup	cm	Cursor motion to row #1 col #2 (G)
cursor_down	cud1	do	Down one line
cursor_home	home	ho	Home cursor (if no **cup**)
cursor_invisible	civis	vi	Make cursor invisible
cursor_left	cub1	le	Move cursor left one space
cursor_mem_address	mrcup	CM	Memory relative cursor addressing (G)
cursor_normal	cnorm	ve	Make cursor appear normal (undo **vs/vi**)
cursor_right	cuf1	nd	Non-destructive space (cursor right)
cursor_to_ll	ll	ll	Last line, first column (if no **cup**)
cursor_up	cuu1	up	Upline (cursor up)
cursor_visible	cvvis	vs	Make cursor very visible
delete_character	dch1	dc	Delete character (*)

delete_line	dl1	dl	Delete line (*)
dis_status_line	dsl	ds	Disable status line
down_half_line	hd	hd	Half-line down (forward 1/2 linefeed)
ena_acs	enacs	eA	Enable alternate char set
enter_alt_charset_mode	smacs	as	Start alternate character set
enter_am_mode	smam	SA	Turn on automatic margins

Variable	**Capname**	**Termcap Code**	**Description**
enter_blink_mode	blink	mb	Turn on blinking
enter_bold_mode	bold	md	Turn on bold (extra bright) mode
enter_ca_mode	smcup	ti	String to begin programs that use **cup**
enter_delete_mode	smdc	dm	Delete mode (enter)
enter_dim_mode	dim	mh	Turn on half-bright mode
enter_insert_mode	smir	im	Insert mode (enter)
enter_protected_mode	prot	mp	Turn on protected mode
enter_reverse_mode	rev	mr	Turn on reverse video mode
enter_secure_mode	invis	mk	Turn on blank mode (chars invisible)
enter_standout_mode	smso	so	Begin standout mode
enter_underline_mode	smul	us	Start underscore mode
enter_xon_mode	smxon	SX	Turn on xon/xoff handshaking
erase_chars	ech	ec	Erase #1 characters (G)
exit_alt_charset_mode	rmacs	ae	End alternate character set
exit_am_mode	rmam	RA	Turn off automatic margins
exit_attribute_mode	sgr0	me	Turn off all attributes
exit_ca_mode	rmcup	te	String to end programs that use **cup**
exit_delete_mode	rmdc	ed	End delete mode
exit_insert_mode	rmir	ei	End insert mode
exit_standout_mode	rmso	se	End standout mode
exit_underline_mode	rmul	ue	End underscore mode
exit_xon_mode	rmxon	RX	Turn off xon/xoff handshaking
flash_screen	flash	vb	Visible bell (may not move cursor)
form_feed	ff	ff	Hardcopy terminal page eject (*)
from_status_line	fsl	fs	Return from status line
init_1string	is1	i1	Terminal initialization string
init_2string	is2	is	Terminal initialization string
init_3string	is3	i3	Terminal initialization string

Variable	Capname	Termcap Code	Description
init_file	if	if	Name of initialization file containing **is**
init_prog	iprog	iP	Path name of program for init
insert_character	ich1	ic	Insert character
insert_line	il1	al	Add new blank line (*)
insert_padding	ip	ip	Insert pad after character inserted (*)
key_a1	ka1	K1	KEY_A1, 0534, Upper left of keypad
key_a3	ka3	K3	KEY_A3, 0535, Upper right of keypad
key_b2	kb2	K2	KEY_B2, 0536, Center of keypad
key_backspace	kbs	kb	KEY_BACKSPACE, 0407, Sent by backspace key
key_beg	kbeg	@1	KEY_BEG, 0542, Sent by beg(inning) key
key_btab	kcbt	kB	KEY_BTAB, 0541, Sent by back-tab key
key_c1	kc1	K4	KEY_C1, 0537, Lower left of keypad
key_c3	kc3	K5	KEY_C3, 0540, Lower right of keypad
key_cancel	kcan	@2	KEY_CANCEL, 0543, Sent by cancel key
key_catab	ktbc	ka	KEY_CATAB, 0526, Sent by clear-all-tabs key
key_clear	kclr	kC	KEY_CLEAR, 0515, Sent by clear-screen or erase key
key_close	kclo	@3	KEY_CLOSE, 0544, Sent by close key
key_command	kcmd	@4	KEY_COMMAND, 0545, Sent by cmd (command) key
key_copy	kcpy	@5	KEY_COPY, 0546, Sent by copy key
key_create	kcrt	@6	KEY_CREATE, 0547, Sent by create key
key_ctab	kctab	kt	KEY_CTAB, 0525, Sent by clear-tab key
key_dc	kdch1	kD	KEY_DC, 0512, Sent by delete-character key
key_dl	kdl1	kL	KEY_DL, 0510, Sent by delete-line key
key_down	kcud1	kd	KEY_DOWN, 0402, Sent by terminal down-arrow key
key_eic	krmir	kM	KEY_EIC, 0514, Sent by **rmir** or **smir** in insert mode
key_end	kend	@7	KEY_END, 0550, Sent by end key

Variable	Capname	Termcap Code	Description
key_enter	kent	@8	KEY_ENTER, 0527, Sent by enter/send key
key_eol	kel	kE	KEY_EOL, 0517, Sent by clear-to-end-of-line key
key_eos	ked	kS	KEY_EOS, 0516, Sent by clear-to-end-of-screen key
key_exit	kext	@9	KEY_EXIT, 0551, Sent by exit key
key_f0	kf0	k0	KEY_F(0), 0410, Sent by function key f0
key_f1	kf1	k1	KEY_F(1), 0411, Sent by function key f1
key_f2	kf2	k2	KEY_F(2), 0412, Sent by function key f2
key_f3	kf3	k3	KEY_F(3), 0413, Sent by function key f3
key_f4	kf4	k4	KEY_F(4), 0414, Sent by function key f4
key_f5	kf5	k5	KEY_F(5), 0415, Sent by function key f5
key_f6	kf6	k6	KEY_F(6), 0416, Sent by function key f6
key_f7	kf7	k7	KEY_F(7), 0417, Sent by function key f7
key_f8	kf8	k8	KEY_F(8), 0420, Sent by function key f8
key_f9	kf9	k9	KEY_F(9), 0421, Sent by function key f9
key_f10	kf10	k;	KEY_F(10), 0422, Sent by function key f10
key_f11	kf11	F1	KEY_F(11), 0423, Sent by function key f11
key_f12	kf12	F2	KEY_F(12), 0424, Sent by function key f12
key_f13	kf13	F3	KEY_F(13), 0425, Sent by function key f13
key_f14	kf14	F4	KEY_F(14), 0426, Sent by function key f14
key_f15	kf15	F5	KEY_F(15), 0427, Sent by function key f15
key_f16	kf16	F6	KEY_F(16), 0430, Sent by function key f16
key_f17	kf17	F7	KEY_F(17), 0431, Sent by function key f17
key_f18	kf18	F8	KEY_F(18), 0432, Sent by function key f18
key_f19	kf19	F9	KEY_F(19), 0433, Sent by function key f19
key_f20	kf20	FA	KEY_F(20), 0434, Sent by function key f20
key_f21	kf21	FB	KEY_F(21), 0435, Sent by function key f21
key_f22	kf22	FC	KEY_F(22), 0436, Sent by function key f22
key_f23	kf23	FD	KEY_F(23), 0437, Sent by function key f23
key_f24	kf24	FE	KEY_F(24), 0440, Sent by function key f24
key_f25	kf25	FF	KEY_F(25), 0441, Sent by function key f25
key_f26	kf26	FG	KEY_F(26), 0442, Sent by function key f26
key_f27	kf27	FH	KEY_F(27), 0443, Sent by function key f27
key_f28	kf28	FI	KEY_F(28), 0444, Sent by function key f28
key_f29	kf29	FJ	KEY_F(29), 0445, Sent by function key f29
key_f30	kf30	FK	KEY_F(30), 0446, Sent by function key f30

Variable	Capname	Termcap Code	Description
key_f31	kf31	FL	KEY_F(31), 0447, Sent by function key f31
key_f32	kf32	FM	KEY_F(32), 0450, Sent by function key f32
key_f33	kf33	FN	KEY_F(13), 0451, Sent by function key f13
key_f34	kf34	FO	KEY_F(34), 0452, Sent by function key f34
key_f35	kf35	FP	KEY_F(35), 0453, Sent by function key f35
key_f36	kf36	FQ	KEY_F(36), 0454, Sent by function key f36
key_f37	kf37	FR	KEY_F(37), 0455, Sent by function key f37
key_f38	kf38	FS	KEY_F(38), 0456, Sent by function key f38
key_f39	kf39	FT	KEY_F(39), 0457, Sent by function key f39
key_f40	kf40	FU	KEY_F(40), 0460, Sent by function key f40
key_f41	kf41	FV	KEY_F(41), 0461, Sent by function key f41
key_f42	kf42	FW	KEY_F(42), 0462, Sent by function key f42
key_f43	kf43	FX	KEY_F(43), 0463, Sent by function key f43
key_f44	kf44	FY	KEY_F(44), 0464, Sent by function key f44
key_f45	kf45	FZ	KEY_F(45), 0465, Sent by function key f45
key_f46	kf46	Fa	KEY_F(46), 0466, Sent by function key f46
key_f47	kf47	Fb	KEY_F(47), 0467, Sent by function key f47
key_f48	kf48	Fc	KEY_F(48), 0470, Sent by function key f48
key_f49	kf49	Fd	KEY_F(49), 0471, Sent by function key f49
key_f50	kf50	Fe	KEY_F(50), 0472, Sent by function key f50
key_f51	kf51	Ff	KEY_F(51), 0473, Sent by function key f51
key_f52	kf52	Fg	KEY_F(52), 0474, Sent by function key f52
key_f53	kf53	Fh	KEY_F(53), 0475, Sent by function key f53
key_f54	kf54	Fi	KEY_F(54), 0476, Sent by function key f54
key_f55	kf55	Fj	KEY_F(55), 0477, Sent by function key f55
key_f56	kf56	Fk	KEY_F(56), 0500, Sent by function key f56
key_f57	kf57	Fl	KEY_F(57), 0501, Sent by function key f57
key_f58	kf58	Fm	KEY_F(58), 0502, Sent by function key f58
key_f59	kf59	Fn	KEY_F(59), 0503, Sent by function key f59
key_f60	kf60	Fo	KEY_F(60), 0504, Sent by function key f60
key_f61	kf61	Fp	KEY_F(61), 0505, Sent by function key f61
key_f62	kf62	Fq	KEY_F(62), 0506, Sent by function key f62
key_f63	kf63	Fr	KEY_F(63), 0507, Sent by function key f63
key_find	kfnd	@0	KEY_FIND, 0552, Sent by find key
key_help	khlp		1
key_home	khome	kh	KEY_HOME, 0406, Sent by home key
key_ic	kich1	kI	KEY_IC, 0513, Sent by ins-char/enter ins-mode key

Variable	Capname	Termcap Code	Description
key_il	kil1	kA	KEY_IL, 0511, Sent by insert-line key
key_left	kcub1	kl	KEY_LEFT, 0404, Sent by terminal left-arrow key
key_ll	kll	kH	KEY_LL, 0533, Sent by home-down key
key_mark	kmrk	%2	KEY_MARK, 0554, Sent by mark key
key_message	kmsg	%3	KEY_MESSAGE, 0555, Sent by message key
key_move	kmov	%4	KEY_MOVE, 0556, Sent by move key
key_next	knxt	%5	KEY_NEXT, 0557, Sent by next-object key
key_npage	knp	kN	KEY_NPAGE, 0522, Sent by next-page key
key_open	kopn	%6	KEY_OPEN, 0560, Sent by open key
key_options	kopt	%7	KEY_OPTIONS, 0561, Sent by options key
key_ppage	kpp	kP	KEY_PPAGE, 0523, Sent by previous-page key
key_previous	kprv	%8	KEY_PREVIOUS, 0562, Sent by previous-object key
key_print	kprt	%9	KEY_PRINT, 0532, Sent by print or copy key
key_redo	krdo	%0	KEY_REDO, 0563, Sent by redo key
key_reference	kref	&1	KEY_REFERENCE, 0564, Sent by ref(erence) key
key_refresh	krfr	&2	KEY_REFRESH, 0565, Sent by refresh key
key_replace	krpl	&3	KEY_REPLACE, 0566, Sent by replace key
key_restart	krst	&4	KEY_RESTART, 0567, Sent by restart key
key_resume	kres	&5	KEY_RESUME, 0570, Sent by resume key
key_right	kcuf1	kr	KEY_RIGHT, 0405, Sent by terminal right-arrow key
key_save	ksav	&6	KEY_SAVE, 0571, Sent by save key

Variable	Capname	Termcap Code	Description
key_sbeg	kBEG	&9	KEY_SBEG, 0572, Sent by shifted beginning key
key_scancel	kCAN	&0	KEY_SCANCEL, 0573, Sent by shifted cancel key
key_scommand	kCMD	*1	KEY_SCOMMAND, 0574, Sent by shifted command key
key_scopy	kCPY	*2	KEY_SCOPY, 0575, Sent by shifted copy key
key_screate	kCRT	*3	KEY_SCREATE, 0576, Sent by shifted create key
key_sdc	kDC	*4	KEY_SDC, 0577, Sent by shifted delete-char key
key_sdl	kDL	*5	KEY_SDL, 0600, Sent by shifted delete-line key
key_select	kslt	*6	KEY_SELECT, 0601, Sent by select key
key_send	kEND	*7	KEY_SEND, 0602, Sent by shifted end key
key_seol	kEOL	*8	KEY_SEOL, 0603, Sent by shifted clear-line key
key_sexit	kEXT	*9	KEY_SEXIT, 0604, Sent by shifted exit key
key_sf	kind	kF	KEY_SF, 0520, Sent by scroll-forward/down key
key_sfind	kFND	*0	KEY_SFIND, 0605, Sent by shifted find key
key_shelp	kHLP	#1	KEY_SHELP, 0606, Sent by shifted help key
key_shome	kHOM	#2	KEY_SHOME, 0607, Sent by shifted home key
key_sic	kIC	#3	KEY_SIC, 0610, Sent by shifted input key
key_sleft	kLFT	#4	KEY_SLEFT, 0611, Sent by shifted left-arrow key
key_smessage	kMSG	%a	KEY_SMESSAGE, 0612, Sent by shifted message key
key_smove	kMOV	%b	KEY_SMOVE, 0613, Sent by shifted move key

Variable	Capname	Termcap Code	Description
key_snext	kNXT	%c	KEY_SNEXT, 0614, Sent by shifted next key
key_soptions	kOPT	%d	KEY_SOPTIONS, 0615, Sent by shifted options key
key_sprevious	kPRV	%e	KEY_SPREVIOUS, 0616, Sent by shifted prev key
key_sprint	kPRT	%f	KEY_SPRINT, 0617, Sent by shifted print key
key_sr	kri	kR	KEY_SR, 0521, Sent by scroll-backward/up key
key_sredo	kRDO	%g	KEY_SREDO, 0620, Sent by shifted redo key
key_sreplace	kRPL	%h	KEY_SREPLACE, 0621, Sent by shifted replace key
key_sright	kRIT	%i	KEY_SRIGHT, 0622, Sent by shifted right-arrow key
key_srsume	kRES	%j	KEY_SRSUME, 0623, Sent by shifted resume key
key_ssave	kSAV	!1	KEY_SSAVE, 0624, Sent by shifted save key
key_ssuspend	kSPD	!2	KEY_SSUSPEND, 0625, Sent by shifted suspend key
key_stab	khts	kT	KEY_STAB, 0524, Sent by set-tab key
key_sundo	kUND	!3	KEY_SUNDO, 0626, Sent by shifted undo key
key_suspend	kspd	&7	KEY_SUSPEND, 0627, Sent by suspend key
key_undo	kund	&8	KEY_UNDO, 0630, Sent by undo key
key_up	kcuu1	ku	KEY_UP, 0403, Sent by terminal up-arrow key
keypad_local	rmkx	ke	Out of ''keypad-transmit'' mode
keypad_xmit	smkx	ks	Put terminal in ''keypad-transmit'' mode
lab_f0	lf0	l0	Labels on function key f0 if not f0
lab_f1	lf1	l1	Labels on function key f1 if not f1

Variable	Capname	Termcap Code	Description
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
ld_upleft	ldul	TL	Upper left corner box character †
ld_upright	ldur	TR	Upper right corner box character †
ld_botleft	ldul	BL	Bottom left corner box character †
ld_botright	ldbl	BR	Bottom right corner box character †
ld_vertleft	ldvl	VL	Left-hand side box character †
ld_vertright	ldvr	VR	Right-hand side box character †
ld_hortop	ldht	TH	Top side box character †
ld_horbot	ldhb	BH	Bottom horizontal box character †
meta_off	rmm	mo	Turn off "meta mode"
meta_on	smm	mm	Turn on "meta mode" (8th bit)
newline	nel	nw	Newline (behaves like **cr** followed by **lf**)
pad_char	pad	pc	Pad character (rather than null)
parm_dch	dch	DC	Delete #1 chars (G*)
parm_delete_line	dl	DL	Delete #1 lines (G*)
parm_down_cursor	cud	DO	Move cursor down #1 lines (G*)
parm_ich	ich	IC	Insert #1 blank chars (G*)

† These CTIX line drawing and video attribute functions are retained from the previous release of CTIX for backward compatibility. AT&T has provided a group of equivalent line drawing functions called **acsc**, documented in the section **Line Graphics** in this manual page. **ctsgr** is provided for backward compatibility: AT&T provides a group of equivalent video attribute functions called **sgr**. The AT&T functions are recommended except when backward compatibility is required.

Variable	Capname	Termcap Code	Description
parm_index	indn	SF	Scroll forward #1 lines (G)
parm_insert_line	il	AL	Add #1 new blank lines (G*)
parm_left_cursor	cub	LE	Move cursor left #1 spaces (G)
parm_right_cursor	cuf	RI	Move cursor right #1 spaces (G*)
parm_rindex	rin	SR	Scroll backward #1 lines (G)
parm_up_cursor	cuu	UP	Move cursor up #1 lines (G*)
pkey_key	pfkey	pk	Prog funct key #1 to type string #2 (G)
pkey_local	pfloc	pl	Prog funct key #1 to execute string #2 (G)
pkey_xmit	pfx	px	Prog funct key #1 to xmit string #2 (G)
plab_norm	pln	pn	Prog label #1 to show string #2 (G)
print_screen	mc0	ps	Print contents of the screen
prtr_non	mc5p	pO	Turn on the printer for #1 bytes (G)
prtr_off	mc4	pf	Turn off the printer
prtr_on	mc5	po	Turn on the printer
repeat_char	rep	rp	Repeat char #1 #2 times (G*)
req_for_input	rfi	RF	Send next input char (for ptys)
reset_1string	rs1	r1	Reset terminal completely to sane modes
reset_2string	rs2	r2	Reset terminal completely to sane modes
reset_3string	rs3	r3	Reset terminal completely to sane modes
reset_file	rf	rf	Name of file containing reset string
restore_cursor	rc	rc	Restore cursor to position of last sc
row_address	vpa	cv	Vertical position absolute (G)
save_cursor	sc	sc	Save cursor position
scroll_forward	ind	sf	Scroll text up
scroll_reverse	ri	sr	Scroll text down
set_attributes	sgr	sa	Define the video attributes #1-#9 (G)
ctset_attributes	ctsgr	cs	Define the video attributes #1-#7 (G) †
set_left_margin	smgl	ML	Set soft left margin
set_right_margin	smgr	MR	Set soft right margin
set_tab	hts	st	Set a tab in all rows, current column
set_window	wind	wi	Current window is lines #1-#2 cols #3-#4 (G)

Variable	Capname	Termcap Code	Description
tab	ht	ta	Tab to next 8 space hardware tab stop
to_status_line	tsl	ts	Go to status line, col #1 (G)
underline_char	uc	uc	Underscore one char and move past it
up_half_line	hu	hu	Half-line up (reverse 1/2 linefeed)
xoff_character	xoffc	XF	X-off character
xon_character	xonc	XN	X-on character

SAMPLE ENTRY

The following entry, which describes the Concept-100 terminal, is among the more complex entries in the *terminfo* file as of this writing.

```
concept100|c100|concept|c104|c100-4p|concept 100,
        am, db, eo, in, mir, ul, xenl,
        cols#80, lines#24, pb#9600, vt#8,
        bel=^G, blank=\EH, blink=\EC, clear=^L$<2*>,
        cnorm=\Ew, cr=^M$<9>, cub1=^H, cud1=^J,
        cuf1=\E=, cup=\Ea%p1%' '%+%c%p2%' '%+%c,
        cuu1=\E;, cvvis=\EW, dch1=\E^A$<16*>, dim=\EE,
        dl1=\E^B$<3*>, ed=\E^C$<16*>, el=\E^U$<16>,
        flash=\Ek$<20>\EK, ht=\t$<8>, il1=\E^R$<3*>,
        ind=^J, .ind=^J$<9>, ip=$<16*>,
        is2=\EU\Ef\E7\E5\E8\El\ENH\EK\E\0\Eo&\0\Eo\47\E,
        kbs=^h, kcub1=\E>, kcud1=\E<, kcuf1=\E=,
        kcuu1=\E;, kf1=\E5, kf2=\E6, kf3=\E7, khome=\E?,
        prot=\EI, rep=\Er%p1%c%p2%' '%+%c$<.2*>,
        rev=\ED, rmcup=\Ev\s\s\s\s$<6>\Ep\r\n,
        rmir=\E\0, rmkx=\Ex, rmso=\Ed\Ee, rmul=\Eg,
        rmul=\Eg, sgr0=\EN\0, smcup=\EU\Ev\s\s8p\Ep\r,
        smir=\E^P, smkx=\EX, smso=\EE\ED, smul=\EG,
```

Entries can continue onto multiple lines by placing white space at the beginning of each line except the first. Lines beginning with "#" are taken as comment lines. Capabilities in *terminfo* are of three types: boolean capabilities which indicate that the terminal has some particular feature, numeric capabilities giving the size of the terminal or particular features, and string capabilities, which give a sequence which can be used to perform particular terminal operations.

Types of Capabilities

All capabilities have names. For instance, the fact that the Concept has *automatic margins* (that is, an automatic return and linefeed when the end of a line is reached) is indicated by the capability **am**. Hence, the description of the

Concept includes **am**. Numeric capabilities are followed by the character # and then the value. Thus, **cols**, which indicates the number of columns the terminal has, gives the value **80** for the Concept. The value can be specified in decimal, octal, or hexadecimal using normal C conventions.

Finally, string-valued capabilities, such as **el** (clear to end of line sequence) are given by the two- to five-character capname, an =, and then a string ending at the next following comma. A delay in milliseconds may appear anywhere in such a capability, enclosed in **$<..>** brackets, as in **el=\EK$<3>**, and padding characters are supplied by **tputs**( ) (see *curses*(3X)) to provide this delay. The delay can be either a number, for example, **20**, or a number followed by an * (that is, **3***), a '/' (that is, **5/**), or both (that is, **10*/**). A '*' indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. (In the case of insert character, the factor is still the number of lines affected. This is always one unless the terminal has **in** and the software uses it.) When a '*' is specified, it is sometimes useful to give a delay of the form **3.5** to specify a delay per unit to tenths of milliseconds. (Only one decimal place is allowed.) A / indicates that the padding is mandatory. Otherwise, if the terminal has **xon** defined, the padding information is advisory and will only be used for cost estimates or when the terminal is in raw mode. Mandatory padding will be transmitted regardless of the setting of **xon**.

A number of escape sequences are provided in the string valued capabilities for easy encoding of characters there. 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 terminals.) Finally, characters can 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 terminal description is by imitating the description of a similar terminal in *terminfo* and to build up a description gradually, using partial descriptions with **vi**(1) to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the *terminfo* file to describe it or the inability of **vi**(1) to work with that terminal. To test a new terminal description, set the environment variable **TERMINFO** to a pathname of a directory containing the compiled description you are working on and programs will look there rather than in */usr/lib/terminfo*. To get the padding for insert-line correct (if the terminal manufacturer did not document it) a severe test is to comment out **xon**, edit a large file at 9600 baud with **vi**(1), delete 16 or so lines from the middle of the screen, then hit 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.

Basic Capabilities

The number of columns on each line for the terminal is given by the **cols** numeric capability. If the terminal has a screen, then the number of lines on the screen is given by the **lines** capability. If the terminal wraps around to the beginning of the next line when it reaches the right margin, then it should have the **am** capability. If the terminal can clear its screen, leaving the cursor in the home position, then this is given by the **clear** string capability. If the terminal overstrikes (rather than clearing a position when a character is struck over) then it should have the **os** capability. If the terminal is a printing terminal, with no soft copy unit, give it both **hc** and **os**. (**os** applies to storage scope terminals, such as Tektronix 4010 series, as well as hard-copy and APL terminals.) If there is a code to move the cursor to the left edge of the current row, give this as **cr**. (Normally this will be carriage return, Control-m.) If there is a code to produce an audible signal (bell, beep, etc) give this as **bel**. If the terminal uses the xon-xoff flow-control protocol, like most terminals, specify **xon**.

If there is a code to move the cursor one position to the left (such as backspace) that capability should be given as **cub1**. Similarly, codes to move to the right, up, and down should be given as **cuf1**, **cuu1**, and **cud1**. These local cursor motions should not alter the text they pass over; for example, you would not normally use ``**cuf1=\s**'' because the space would erase the character moved over.

A very important point here is that the local cursor motions encoded in *terminfo* are undefined at the left and top edges of a screen terminal. Programs should never attempt to backspace around the left edge, unless **bw** is given, and should never attempt to go up locally off the top. In order to scroll text up, a program will go to the bottom left corner of the screen and send the **ind** (index) string.

To scroll text down, a program goes to the top left corner of the screen and sends the **ri** (reverse index) string. The strings **ind** and **ri** are undefined when not on their respective corners of the screen.

Parameterized versions of the scrolling sequences are **indn** and **rin** which have the same semantics as **ind** and **ri** except that they take one parameter, and scroll that many lines. They are also undefined except at the appropriate edge of the screen.

The **am** capability tells whether the cursor sticks at the right edge of the screen when text is output, but this does not necessarily apply to a **cuf1** from the last column. The only local motion which is defined from the left edge is if **bw** is given, then a **cub1** from the left edge will move to the right edge of the previous row. If **bw** is not given, the effect is undefined. This is useful for drawing a box around the edge of the screen, for example. If the terminal has switch selectable automatic margins, the *terminfo* file usually assumes that this is on; that is, **am**. If the terminal has a command which moves to the first column of the next line, that command can be given as **nel** (newline). It does not matter if the command clears the remainder of the current line, so if the terminal has no **cr** and **lf** it may still be possible to craft a working **nel** out of one or both of them.

These capabilities suffice to describe hardcopy and screen terminals. Thus, the model 33 teletype is described as:

```
33|tty33|tty|model 33 teletype,
        bel=^G, cols#72, cr=^M, cud1=^J, hc, ind=^J, os,
```

while the Lear Siegler ADM-3 is described as

```
adm3|lsi adm3,
        am, bel=^G, clear=^Z, cols#80, cr=^M,
        cub1=^H, cud1=^J, ind=^J, lines#24,
```

Parameterized Strings

Cursor addressing and other strings requiring parameters in the terminal are described by a parameterized string capability, with **printf**(3S)-like escapes (%**x**) in it. For example, to address the cursor, the **cup** capability is given, using two parameters: the row and column to address to. (Rows and columns are numbered from zero and refer to the physical screen visible to the user, not to any unseen memory.) If the terminal has memory relative cursor addressing, that can be indicated by **mrcup**.

The parameter mechanism uses a stack and special % codes to manipulate it in the manner of a Reverse Polish Notation (postfix) calculator. Typically, a sequence will push one of the parameters onto the stack and then print it in some format. Often more complex operations are necessary. Binary operations are in postfix form with the operands in the usual order. That is, to get x-5 one would use **%gx%{5}%-**.

The % encodings have the following meanings:

%%	outputs '%'
%[[:]*flags*][*width*[*.precision*]][**doxXs**]	as in printf, flags are [- + #] and space
%c	print pop( ) gives %c
%p[1-9]	push i^{th} parm
%P[a-z]	set variable [a-z] to pop( )
%g[a-z]	get variable [a-z] and push it
%'*c*'	push char constant *c*
%{*nn*}	push decimal constant *nn*
%l	push strlen(pop( ))
%+ %- %* %/ %m	arithmetic (%m is mod): push(pop( ) op pop( ))
%& %| %^	bit operations: push(pop( ) op pop( ))
%= %> %<	logical operations: push(pop( ) op pop( ))
%A %O	logical operations: and, or
%! %~	unary operations: push(op pop( ))
%i	(for ANSI terminals) add 1 to first parm, if one parm present, or first two parms, if more than one parm present
%? expr %t thenpart %e elsepart %;	if-then-else, %e elsepart is optional; else-if's are possible 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, .e.g "%:-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%2.2d Y$<6>".

The Micro-Term ACT-IV needs the current row and column sent preceded by a **^T**, with the row and column simply encoded in binary, "**cup=**^T%p1%c%p2%c". Terminals which use "%c" need to be able to backspace the cursor (**cub1**), and to move the cursor up one line on the screen (**cuu1**). This is necessary because it is not always safe to transmit **\n**, **^D**, and **\r**, as the system may change or discard them. (The library routines dealing with *terminfo* set tty modes so that 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: "**cup=**\E=%p1%'\s'%+%c%p2%'\s'%+%c". After sending "\E=", this pushes the first parameter, pushes the ASCII value for a space (32), adds them (pushing the sum on the stack in place of the two previous values), and outputs that value as a character. Then the same is done for the second parameter. More complex arithmetic is possible using the stack.

Cursor Motions

If the terminal has a fast way to home the cursor (to very upper left corner of screen) then this can be given as **home**; similarly a fast way of getting to the lower left-hand corner can be given as **ll**; this may involve going up with **cuu1** from the home position, but a program should never do this itself (unless **ll** does) because it can make no assumption about the effect of moving up from the home position. Note 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 terminal has row or column absolute-cursor addressing, these can be given as single parameter capabilities **hpa** (horizontal position absolute) and **vpa** (vertical position absolute). Sometimes these are shorter than the more general two-parameter sequence (as with the Hewlett-Packard 2645) and can be used in preference to **cup**. If there are parameterized local motions (for example, move *n* spaces to the right) these can be given as **cud**, **cub**, **cuf**, and **cuu** with a single parameter indicating how many spaces to move. These are primarily useful if the terminal does not have **cup**, such as the Tektronix 4025.

Area Clears

If the terminal can clear from the current position to the end of the line, leaving the cursor where it is, this should be given as **el**. If the terminal can clear from the beginning of the line to the current position inclusive, leaving the cursor where it is, this should be given as **el1**. If the terminal can clear from the current position to the end of the display, then this should be given as **ed**. **ed** is only defined from the first column of a line. (Thus, it can be simulated by a request to delete a large number of lines, if a true **ed** is not available.)

Insert/delete line

If the terminal can open a new blank line before the line where the cursor is, this should be given as **il1**; this is done only from the first position of a line. The cursor must then appear on the newly blank line. If the terminal can delete the line which the cursor is on, then this should be given as **dl1**; this is done only from the first position on the line to be deleted. Versions of **il1** and **dl1** which take a single parameter and insert or delete that many lines can be given as **il** and **dl**.

If the terminal has a settable destructive scrolling region (like the VT100) the command to set this can be described with the **csr** capability, which takes two parameters: the top and bottom lines of the scrolling region. The cursor position is, alas, undefined after using this command. It is possible to get the effect of insert or delete line using this command—the **sc** and **rc** (save and restore cursor) commands are also useful. Inserting lines at the top or bottom of the screen can also be done using **ri** or **ind** on many terminals without a true insert/delete line, and is often faster even on terminals with those features.

To determine whether a terminal has destructive scrolling regions or non-destructive scrolling regions, create a scrolling region in the middle of the screen, place data on the bottom line of the scrolling region, move the cursor to the top line of the scrolling region, and do a reverse index (**ri**) followed by a delete line (**dl1**) or index (**ind**). If the data that was originally on the bottom line of the scrolling region was restored into the scrolling region by the **dl1** or **ind**, then the terminal has non-destructive scrolling regions. Otherwise, it has destructive scrolling regions. Do not specify **csr** if the terminal has non-destructive scrolling regions, unless **ind**, **ri**, **indn**, **rin**, **dl**, and **dl1** all simulate destructive scrolling.

If the terminal has the ability to define a window as part of memory, which all commands affect, it should be given as the parameterized string **wind**. The four parameters are the starting and ending lines in memory and the starting and ending columns in memory, in that order.

If the terminal can retain display memory above, then the **da** capability should be given; if display memory can be retained below, then **db** should be given. These indicate that deleting a line or scrolling a full screen may bring non-blank lines up from below or that scrolling back with **ri** may bring down non-blank lines.

Insert/Delete Character

There are two basic kinds of intelligent terminals with respect to insert/delete character operations which can be described using *terminfo*. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting upon an insert or delete only to an untyped blank on the screen which is either eliminated, or expanded to two untyped blanks. You can determine the kind of terminal you have by clearing the screen and then typing text separated by cursor motions. Type ``**abc def**'' using local cursor motions (not 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 which have an insert mode and terminals which send a simple sequence to open a blank position on the current line. Give as **smir** the sequence to get into insert mode. Give as **rmir** the sequence to leave insert mode. Now give as **ich1** any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give **ich1**; terminals which send a sequence to open a screen position should give it here. (If your terminal has both, insert mode is usually preferable to **ich1**. Do not give both unless the terminal actually requires both to be used in combination.) If post-insert padding is needed, give this as a number of milliseconds padding in **ip** (a string option). Any other sequence which may need to be sent after an insert of a single character may also be given in **ip**. If your terminal needs both to be placed into an 'insert mode' and a special code to precede each inserted character, then both **smir/rmir** and **ich1** can be given, and both will be used. The **ich** capability, with one parameter, *n*, will repeat the effects of **ich1** *n* times.

If padding is necessary between characters typed while not in insert mode, give this as a number of milliseconds padding in **rmp**.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (for example, if there is a tab after the insertion position). If your terminal allows motion while in insert mode you can give the capability **mir** to speed up inserting in this case. Omitting **mir** will affect only speed. Some terminals (notably Datamedia's) must not have **mir** because of the way their insert mode works.

Finally, you can specify **dch1** to delete a single character, **dch** with one parameter, *n*, to delete *n* characters, and delete mode by giving **smdc** and **rmdc** to enter and exit delete mode (any mode the terminal needs to be placed in for **dch1** to work).

A command to erase *n* characters (equivalent to outputting *n* blanks without moving the cursor) can be given as **ech** with one parameter.

Highlighting, Underlining, and Visible Bells

If your terminal has one or more kinds of display attributes, these can be represented in a number of different ways. You should choose one display form as *standout mode* (see *curses*(3X)), representing a good, high contrast, easy-on-the-eyes, format for highlighting error messages and other attention getters. (If you have a choice, reverse-video plus half-bright is good, or reverse-video alone; however, different users have different preferences on different terminals.) The sequences to enter and exit standout mode are given as **smso** and **rmso**, respectively. If the code to change into or out of standout mode leaves one or even two 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.

Codes to begin underlining and end underlining can be given as **smul** and **rmul** respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Micro-Term MIME, this can be given as **uc**.

Other capabilities to enter various highlighting modes include **blink** (blinking), **bold** (bold or extra-bright), **dim** (dim or half-bright), **invis** (blanking or invisible text), **prot** (protected), **rev** (reverse-video), **sgr0** (turn off all attribute modes), **smacs** (enter alternate-character-set mode), and **rmacs** (exit alternate-character-set mode). Turning on any of these modes singly may or may not turn off other modes. If a command is necessary before alternate character set mode is entered, give the sequence in **enacs** (enable alternate-character-set mode).

If there is a sequence to set arbitrary combinations of modes, this should be given as **sgr** (set attributes), taking nine parameters. Each parameter is either **0** or non-zero, as the corresponding attribute is on or off. The nine parameters are, in order: standout, underline, reverse, blink, dim, bold, blank, protect, alternate character set. Not all modes need be supported by **sgr**, only those for which corresponding separate attribute commands exist. (See the example at the end of this section.)

Terminals with the "magic cookie" glitch (**xmc**) deposit special "cookies" when they receive mode-setting sequences, which affect the display algorithm rather than having extra bits for each character. Some terminals, such as the Hewlett-Packard 2621, automatically leave standout mode when they move to a new line or the cursor is addressed. Programs using standout mode should exit standout mode before moving the cursor or sending a newline, unless the **msgr** capability, asserting that it is safe to move in standout mode, is present.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement), then this can be given as **flash**; it must not move the cursor. A good flash can be done by changing the screen into reverse video, pad for 200 ms, then return the screen to normal video.

If the cursor needs to be made more visible than normal when it is not on the bottom line (to make, for example, a non-blinking underline into an easier to find block or blinking underline) give this sequence as **cvvis**. The boolean **chts** should also be given. If there is a way to make the cursor completely invisible, give that as **civis**. The capability **cnorm** should be given which undoes the effects of either of these modes.

If the terminal needs to be in a special mode when running a program that uses these capabilities, the codes to enter and exit this mode can be given as **smcup** and **rmcup**. This arises, for example, from terminals like the Concept with more than one page of memory. If the terminal has only memory relative cursor addressing and not screen relative cursor addressing, a one screen-sized window must be fixed into the terminal for cursor addressing to work properly. This is also used for the Tektronix 4025, where **smcup** sets the command character to be the one used by **terminfo**. If the **smcup** sequence will not restore the screen after an **rmcup** sequence is output (to the state prior to outputting **rmcup**), specify **nrrmc**.

If your terminal generates underlined characters by using the underline character (with no special codes needed) even though it does not otherwise overstrike characters, then you should give the capability **ul**. For terminals where a character overstriking another leaves both characters on the screen, give the capability **os**. If overstrikes are erasable with a blank, then this should be indicated by giving **eo**.

Example of highlighting: assume that the terminal under question needs the following escape sequences to turn on various modes.

tparm parameter	*attribute*	*escape sequence*
	none	\E[0m
p1	standout	\E[0;4;7m
p2	underline	\E[0;3m
p3	reverse	\E[0;4m
p4	blink	\E[0;5m
p5	dim	\E[0;7m
p6	bold	\E[0;3;4m
p7	invis	\E[0;8m
p8	protect	not available
p9	altcharset	^O (off) ^N(on)

Note 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, since this terminal has no *bold* mode, *bold* is set up as the combination of *reverse* and *underline*. In addition, to allow combinations, such as *underline+blink*, the sequence to use would be **\E[0;3;5m**. The terminal 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%;,
```

Keypad and Function Keys

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. Note that it is not possible to handle terminals where the keypad only works in local (this applies, for example, to the unshifted Hewlett-Packard 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as **smkx** and **rmkx**. Otherwise the keypad is assumed to always transmit.

The codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys can be given as **kcub1, kcuf1, kcuu1, kcud1,** and **khome** respectively. If there are function keys such as f0, f1, ..., f63, the codes they send can be given as **kf0, kf1, ..., kf63.** If the first 11 keys have labels other than the default f0 through f10, the labels can be given as **lf0, lf1, ..., lf10.** The codes transmitted by certain other special keys can be given: **kll** (home down), **kbs** (backspace), **ktbc** (clear all 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 given as **pfkey, pfloc,** and **pfx.** A string to program their soft-screen labels can be given as **pln.** Each of these strings takes two parameters: the function key number to program and the string to program it with. The difference between the capabilities is that **pfkey** causes pressing the given key to be the same as the user typing the given string; **pfloc** 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 how many soft labels there are and their width and height. If there are commands to turn the labels on and off, give them in **smln** and **rmln. smln** is normally output after one or more **pln** sequences to make sure that the change becomes visible.

Tabs and Initialization

If the terminal has hardware tabs, the command to advance to the next tab stop can be given as **ht** (usually Control-i). A "backtab" command which moves leftward to the next tab stop can be given as **cbt.** By convention, if the teletype modes indicate that tabs are being expanded by the computer rather than being sent to the terminal, programs should not use **ht** or **cbt** even if they are present, since the user may not have the tab stops properly set. If the terminal has hardware tabs which are initially set every *n* spaces when the terminal is powered up, the numeric parameter **it** is given, showing the number of spaces the 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 terminal has tab stops that can be saved in nonvolatile memory, the *terminfo* description can assume that they are properly set. If there are commands to set and clear tab stops, they can be given as **tbc** (clear all tab stops) and **hts** (set a tab stop in the current column of every row).

Other capabilities include: **is1**, **is2**, and **is3**, initialization strings for the terminal; **iprog**, the path name of a program to be run to initialize the terminal; and **if**, the name of a file containing long initialization strings. These strings are expected to set the terminal into modes consistent with the rest of the *terminfo* description. They must be sent to the terminal each time the user logs in and be output in the following order: run the program **iprog**; output **is1**; output **is2**; set the margins using **mgc**, **smgl** and **smgr**; set the tabs using **tbc** and **hts**; print the file **if**; and finally output **is3**. This is usually done using the **init** option of *tput*(1); see *profile*(4).

Most initialization is done with **is2**. Special terminal modes can be set up without duplicating strings by putting the common sequences in **is2** and special cases in **is1** and **is3**. Sequences that do a harder reset from a totally unknown state can be given as **rs1**, **rs2**, **rf**, and **rs3**, analogous to **is1**, **is2**, **is3**, and **if**. (The method using files, **if** and **rf**, is used for a few terminals, from */usr/lib/tabset/**; however, the recommended method is to use the initialization and reset strings.) These strings are output by **tput reset**, which is used when the terminal gets into a wedged state. Commands are normally placed in **rs1**, **rs2**, **rs3**, and **rf** only if they produce annoying effects on the screen and are not necessary when logging in. For example, the command to set a terminal into 80-column mode would normally be part of **is2**, but on some terminals it causes an annoying glitch on the screen and is not normally needed since the terminal is usually already in 80-column mode.

If a more complex sequence is needed to set the tabs than can be described by using **tbc** and **hts**, the sequence can be placed in **is2** or **if**.

If there are commands to set and clear margins, they can be given as **mgc** (clear all margins), **smgl** (set left margin), and **smgr** (set right margin).

Delays

Certain capabilities control padding in the tty driver [see *termio*(7)]. These are primarily needed by hard-copy terminals, and are used by **tput init** to set tty modes appropriately. Delays embedded in the capabilities **cr**, **ind**, **cub1**, **ff**, and

tab can be used to set the appropriate delay bits to be set in the tty driver. If **pb** (padding baud rate) is given, these values can be ignored at baud rates below the value of **pb**.

Status Lines

If the terminal has an extra "status line" that is not normally used by software, this fact can be indicated. If the status line is viewed as an extra line below the bottom line, into which one can cursor address normally (such as the Heathkit h19's 25th line, or the 24th line of a VT100 which is set to a 23-line scrolling region), the capability **hs** should be given. Special strings that go to a given column of the status line and return from the status line can be given as **tsl** and **fsl**. (**fsl** must leave the cursor position in the same place it was before **tsl**. If necessary, the **sc** and **rc** strings can be included in **tsl** and **fsl** to get this effect.) The capability **tsl** takes one parameter, which is the column number of the status line the cursor is to be moved to.

If escape sequences and other special commands, such as tab, work while in the status line, the flag **eslok** can be given. A string which turns off the status line (or otherwise erases its contents) should be given as **dsl**. If the terminal has commands to save and restore the position of the cursor, give them as **sc** and **rc**. The status line is normally assumed to be the same width as the rest of the screen, for example, **cols**. If the status line is a different width (possibly because the terminal does not allow an entire line to be loaded) the width, in columns, can be indicated with the numeric parameter **wsl**.

Line Graphics

If the terminal has a line drawing alternate character set, the mapping of glyph to character would be given in **acsc**. The definition of this string is based on the alternate character set used in the DEC VT100 terminal, extended slightly with some characters from the AT&T 4410v1 terminal.

glyph name	vt100+ *character*
arrow pointing right	+
arrow pointing left	,
arrow pointing down	.
solid square block	0
lantern symbol	I
arrow pointing up	-
diamond	`
checker board (stipple)	a
degree symbol	f
plus/minus	g
board of squares	h
lower right corner	j
upper right corner	k
upper left corner	l
lower left corner	m
plus	n
scan line 1	o
horizontal line	q
scan line 9	s
left tee (├)	t
right tee (┤)	u
bottom tee (┴)	v
top tee (┬)	w
vertical line	x
bullet	~

The best way to describe a new terminal's line graphics set is to add a third column to the above table with the characters for the new terminal that produce the appropriate glyph when the terminal is in the alternate character set mode.

For example,

glyph name	vt100+ *char*	*new tty char*
upper left corner	l	R
lower left corner	m	F
upper right corner	k	T
lower right corner	j	G
horizontal line	q	,
vertical line	x	.

Now write down the characters left to right, as in "**acsc=lRmFkTjGq\,x.**".

The AT&T functions above are recommended for present and future development; the following CTIX functional equivalents are retained for backward compatibility. Both sets of functions are supported by this release of CTIX .

Eight single-line drawing characters can be given. The eight eight characters that can be specified represent the top left corner, top right corner, bottom left corner, bottom right corner left side, right side, top side, and bottom side of a solid line box. The four corner are specified with **ldul**, **ldur**, **ldbl**, and **ldbr**. The four sides can be specified with **ldvl**, **ldvr**, **ldht**, and **ldhb**. If the terminal must be in a special mode to draw the line characters, specify the necessary sequences to enter and exit the mode as one of the six highlight modes (alternate character set is usually a good choice); then give the mode number as a numeric value to **ldatt**. The correspondence of highlight modes and numeric values is as follows:

attribute	*mode*
1	underline
2	reverse
3	blink
4	dim
5	bold
6	alternate character set
7	standout

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.

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 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 can be given in **smxon** and **rmxon**. If the characters used for handshaking are not ^S and ^Q, they can be specified with **xonc** and **xoffc**.

If the terminal has a "meta key" which acts as a shift key, setting the 8th bit of any character transmitted, this fact can be indicated with **km**. Otherwise, software will assume that the 8th bit is parity and it will usually be cleared. If strings exist to turn this "meta mode" on and off, they can be given as **smm** and **rmm**.

If the terminal has more lines of memory than will fit on the screen at once, the number of lines of memory can be indicated with **lm**. A value of **lm**#0 indicates that the number of lines is not fixed, but that there is still more memory than fits on the screen.

Media copy strings which control an auxiliary printer connected to the terminal can be given as **mc0**: print the contents of the screen, **mc4**: turn off the printer, and **mc5**: turn on the printer. When the printer is on, all text sent to the terminal will be sent to the printer. A variation, **mc5p**, takes one parameter, and leaves the printer on for as many characters as the value of the parameter, then turns the printer off. The parameter should not exceed 255. If the text is not displayed on the terminal screen when the printer is on, specify **mc5i** (silent printer). All text, including **mc4**, is transparently passed to the printer while an **mc5p** is in effect.

Special Cases

The working model used by *terminfo* fits most terminals reasonably well. However, some terminals do not completely match that model, requiring special support by *terminfo*. These are not meant to be construed as deficiencies in the terminals; they are just differences between the working model and the actual hardware. They may be unusual devices or, for some reason, do not have all the features of the *terminfo* model implemented.

Terminals which can not display tilde (~) characters, such as certain Hazeltine terminals, should indicate **hz**.

Terminals which ignore a linefeed immediately after an **am** wrap, such as the Concept 100, should indicate **xenl**. Those terminals whose cursor remains on the right-most column until another character has been received, rather than wrapping immediately upon receiving the right-most character, such as the VT100, should also indicate **xenl**.

If **el** is required to get rid of standout (instead of writing normal text on top of it), **xhp** should be given.

Those Teleray terminals whose tabs turn all characters moved over to blanks, should indicate **xt** (destructive 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 instead necessary 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.

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 can be given.

FILES

/usr/lib/terminfo/?/*	compiled terminal description database
/usr/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

captoinfo(1M), infocmp(1M), tic(1M), tput(1), curses(3X), printf(3S), term(5), termio(7).
UNIX System V Release 3.2 Programmer's Guide.

WARNING

As described in the "Tabs and Initialization" section above, a terminal's initialization strings, **is1**, **is2**, and **is3**, if defined, must be output before a *curses*(3X) program is run. An available mechanism for outputting such strings is **tput init** [see *tput*(1) and *profile*(4)].

NAME

ttytype - list of terminal types by terminal number

DESCRIPTION

/etc/ttytype is a text file that contains, for each terminal configured, the terminal type as described in *termcap*(4). It is used by *tset*(1) when that program sets the **TERM** environment variable.

A line in *ttytype* consists of a terminal name (one of the abbreviations from the first field of the *termcap* entry), followed by a space, followed by the special file name of the terminal without the initial **/dev/**.

EXAMPLES

```
pt tty000
```

FILES

/etc/ttytype

SEE ALSO

tset(1), termcap(4).
S/Series CTIX Administrator's Guide.

NAME

unistd - file header for symbolic constants

SYNOPSIS

#include <unistd.h>

DESCRIPTION

The header file *<unistd.h>* lists the symbolic constants and structures not already defined or declared in some other header file.

```
/* Symbolic constants for the "access" routine: */

#define R_OK      4   /* Test for Read permission */
#define W_OK      2   /* Test for Write permission */
#define X_OK      1   /* Test for eXecute permission */
#define F_OK      0   /* Test for existence of File */

#define F_ULOCK 0    /* Unlock a previously locked region */
#define F_LOCK    1   /* Lock a region for exclusive use */
#define F_TLOCK 2    /* Test and lock a region for exclusive use */
#define F_TEST    3   /* Test a region for other processes locks */

/* Symbolic constants for the "lseek" routine: */

#define SEEK_SET0     /* Set file pointer to "offset" */
#define SEEK_CUR      1/* Set file pointer to current plus "offset" */
#define SEEK_END      2/* Set file pointer to EOF plus "offset" */

/* Pathnames:*/

#define GF_PATH  /etc/group    /*Pathname of the group file */
#define PF_PATH  /etc/passwd  /*Pathname of the passwd file */
```

NAME

utmp, wtmp - utmp and wtmp entry formats

SYNOPSIS

#include <sys/types.h>
#include <utmp.h>

DESCRIPTION

These files, which hold user and accounting information for such commands as *who*(1), *write*(1) and *login*(1), have the following structure as defined by **<utmp.h>**:

```
#define  UTMP_FILE     "/etc/utmp"
#define  WTMP_FILE     "/etc/wtmp"
#define  ut_name       ut_user

struct   utmp {
         char    ut_user[8];   /* User login name */
         char    ut_id[4];     /* /etc/inittab id (usually line #) */
         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;            /* The 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
                         /* Largest legal value of ut_type */
```

```
/* Special strings or formats used in the "ut_line" field */
/* when accounting for something other than a process */
/* No string for the ut_line field can be more than 11 */
/* chars + a NULL in length */
#define   RUNLVL_MSG    "run-level %c"
#define   BOOT_MSG      "system boot"
#define   OTIME_MSG     "old time"
#define   NTIME_MSG     "new time"
```

FILES

/etc/utmp
/etc/wtmp

SEE ALSO

login(1), who(1), write(1), getut(3C).

NAME

intro - introduction to miscellany

DESCRIPTION

This section describes miscellaneous facilities such as macro packages, character set tables, etc.

NAME

Devices - configuration file for uucp communications lines

SYNOPSIS

/usr/lib/uucp/Devices

DESCRIPTION

The **/usr/lib/uucp/Devices** text file contains configuration specifications for communications devices, such as modems or direct lines. Each line in the file describes a single device and how it communicates with a remote system. Comment lines begin with a pound sign (#). The UUCP system uses the **/usr/lib/uucp/Devices** file in conjunction with the **/usr/lib/uucp/Dialers** file to place a call.

Each line containes five or more fields delimited by spaces. The first field is the line type as specified in the **/usr/lib/uucp/Systems** file; for direct lines, the first field is the name of the remote system.

The remaining fields give the device name; the calling device indicator (such as for 801 calling units), if used; the speed, which may be specified as ANY; and the name of the caller as specified in the **/usr/lib/uucp/Dialers** file. The last field, the name of the caller, can be followed by a token format (containing **\D** or **\T**); pairs of these dialer name/token format fields can be repeated if more than one dialer must be used in succession to make the connection. If no token format is specified, a **\D** is used for a dialer name that references the **/usr/lib/uucp/Dialers** file; a **\T** is used for internal dialer types such as 801. Unused fields are replaced by a hyphen (-).

EXAMPLE

The following entry configures a 1200-baud intelligent modem on device contty for use with UUCP:

    ACU contty - 1200 penril

FILES

/usr/lib/uucp/Devices
/usr/lib/uucp/Dialers
/usr/lib/uucp/Systems

SEE ALSO

uucp(1C), dial(3C), Dialers(5).
S/Series CTIX Administrator's Guide.

NAME

Dialers - ACU/modem calling protocols

SYNOPSIS

/usr/lib/uucp/Dialers

DESCRIPTION

Dialers describes the call-placing protocols for intelligent modems, ACUs (automatic calling units), and other serial switched devices such as data switches. When a connection is requested via the UUCP system, CTIX looks for a description of the called system in the **/usr/lib/uucp/Systems** file, where the type of line is specified for connection to that system. CTIX then checks the **/usr/lib/uucp/Devices** file for a description of the line, its speed and its Dialers name. The Dialers name given in the **Devices** file corresponds to the first field of the **Dialers** file.

Dialers is a text file that contains the dialing script for the modems that are configured in the **Devices** file. Each description begins on a new line and has three or more fields, delimited by spaces.

The first field of the description is the name of the modem or device as specified in the **Devices** file.

The second field specifies the codes used by that particular modem for secondary dial tone (=) and pause (-); this field enables CTIX to translate from the standard 801 codes (= and -) to the special characters used by that particular device.

The remaining fields make up the chat script necessary to establish communication with the modem.

The modem chat script is composed of command strings to the modem and response strings expected in return from the modem. The strings consist of ASCII and control characters recognized by the individual modem or device. Spaces delimit the end of a send or receive sequence. The first string is an expect string.

Several modems and switches are already provided in the **Dialers** file. Additional devices can be configured by studying the manufacturers' manuals to determine the appropriate send/receive sequences for other modems.

In the string sequences of the send/receive fields the following escape sequences represent control codes:

\ddd Octal number.

\c Suppress new line (valid only after \r or at the end of a field).

\d Delay (two seconds).

\D Substitute the telephone number (from the **/usr/lib/uucp/Systems** file or *cu*(1C)), without character translation.

\e Turn off echo checking.

\E Turn on echo checking (for slow devices).

\K Insert a BREAK.

\n New-line.

\p Pause (a slight delay of one-quarter to one-half second).

\r Carriage return.

\T Substitute the telephone number (from the **/usr/lib/uucp/Systems** file or *cu*(1C)), with character translation. Character translation interprets the 801 codes in the second field and expands any symbols found in the **/usr/lib/uucp/Dialcodes** file.

Comments delimited by a pound sign (#), spaces, or tabs are ignored. Any line terminated by a backslash (\) continues to the next line.

EXAMPLE

The following example establishes communication with a Ventel modem:

```
ventel =&-%        "" \r\p\r\c $ <K\T%%\r>\c ONLINE!
```

The first field, "ventel," is the name of the modem that corresponds to a "ventel" caller type in the fifth or subsequent field of a **Devices** file entry. The second field describes the modem's convention for the secondary dial tone (&) and a pause (%) command. The remaining fields consist of five strings separated by spaces. The five strings are interpreted as follows:

1. The first expect string ("") is null.
2. Send to the modem a series of carriage returns to elicit a prompt.
3. The modem should respond with a dollar sign ($).
4. Send the telephone number (\T) to the modem.
5. Upon connection the modem should respond with the string 'ONLINE!'.

NAME

Dialers - ACU/modem calling protocols

SYNOPSIS

/usr/lib/uucp/Dialers

DESCRIPTION

Dialers describes the call-placing protocols for intelligent modems, ACUs (automatic calling units), and other serial switched devices such as data switches. When a connection is requested via the UUCP system, CTIX looks for a description of the called system in the **/usr/lib/uucp/Systems** file, where the type of line is specified for connection to that system. CTIX then checks the **/usr/lib/uucp/Devices** file for a description of the line, its speed and its Dialers name. The Dialers name given in the **Devices** file corresponds to the first field of the **Dialers** file.

Dialers is a text file that contains the dialing script for the modems that are configured in the **Devices** file. Each description begins on a new line and has three or more fields, delimited by spaces.

The first field of the description is the name of the modem or device as specified in the **Devices** file.

The second field specifies the codes used by that particular modem for secondary dial tone (=) and pause (-); this field enables CTIX to translate from the standard 801 codes (= and -) to the special characters used by that particular device.

The remaining fields make up the chat script necessary to establish communication with the modem.

The modem chat script is composed of command strings to the modem and response strings expected in return from the modem. The strings consist of ASCII and control characters recognized by the individual modem or device. Spaces delimit the end of a send or receive sequence. The first string is an expect string.

Several modems and switches are already provided in the **Dialers** file. Additional devices can be configured by studying the manufacturers' manuals to determine the appropriate send/receive sequences for other modems.

In the string sequences of the send/receive fields the following escape sequences represent control codes:

\ddd Octal number.

\c Suppress new line (valid only after \r or at the end of a field).

\d Delay (two seconds).

\D Substitute the telephone number (from the **/usr/lib/uucp/Systems** file or *cu*(1C)), without character translation.

\e Turn off echo checking.

\E Turn on echo checking (for slow devices).

\K Insert a BREAK.

\n New-line.

\p Pause (a slight delay of one-quarter to one-half second).

\r Carriage return.

\T Substitute the telephone number (from the **/usr/lib/uucp/Systems** file or *cu*(1C)), with character translation. Character translation interprets the 801 codes in the second field and expands any symbols found in the **/usr/lib/uucp/Dialcodes** file.

Comments delimited by a pound sign (#), spaces, or tabs are ignored. Any line terminated by a backslash (\) continues to the next line.

EXAMPLE

The following example establishes communication with a Ventel modem:

```
ventel =&-%        "" \r\p\r\c $ <K\T%%\r>\c ONLINE!
```

The first field, "ventel," is the name of the modem that corresponds to a "ventel" caller type in the fifth or subsequent field of a **Devices** file entry. The second field describes the modem's convention for the secondary dial tone (&) and a pause (%) command. The remaining fields consist of five strings separated by spaces. The five strings are interpreted as follows:

1. The first expect string ("") is null.
2. Send to the modem a series of carriage returns to elicit a prompt.
3. The modem should respond with a dollar sign ($).
4. Send the telephone number (\T) to the modem.
5. Upon connection the modem should respond with the string 'ONLINE!'.

FILES

/usr/lib/uucp/Devices
/usr/lib/uucp/Dialcodes
/usr/lib/uucp/Systems

SEE ALSO

uucp(1C), dial(3C), Devices(5).
S/Series CTIX Administrator's Guide.

NAME

ascii - map of ASCII character set

SYNOPSIS

cat /usr/pub/ascii

DESCRIPTION

ascii is a map of the ASCII character set, giving both octal and hexadecimal equivalents of each character, to be printed as needed. Entering the command shown in the SYNOPSIS writes the display shown below to the standard output:

```
|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|
```

FILES

/usr/pub/ascii

NAME

environ - user environment

DESCRIPTION

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

CFTIME The default format string to be used by the *date*(1) command and the **ascftime**( ) and **cftime**( ) routines [see *ctime*(3C)]. If **CFTIME** is not set or is null, the default format string specified in the **/lib/cftime/***LANGUAGE* file (if it exists) is used in its place [see *cftime*(4)].

CHRCLASS A value that corresponds to a file in **/lib/chrclass** containing character classification and conversion information. This information is used by commands [such as *cat*(1), *ed*(1), *sort*(1), etc.] to classify characters as alphabetic, printable, uppercase, etc. and to convert characters to uppercase or lowercase.

When a program or command begins execution, the tables containing this information are initialized based on the value of **CHRCLASS**. If **CHRCLASS** is non-existent, null, set to a value for which no file exists in **/lib/chrclass**, or errors occur while reading the file, the ASCII character set is used. During execution, a program or command can change the values in these tables by calling the **setchrclass**( ) routine. For more detail, see *ctype*(3C).

These tables are created using the *chrtbl*(1M) command.

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

LANGUAGE A language for which a printable file by that name exists in **/lib/cftime.** This information is used by commands [such as *date*(1), *ls*(1), *sort*(1), etc.] to print date and time information in the language specified.

If **LANGUAGE** is non-existent, null, set to a value for which no file exists in **/lib/cftime**, or errors occur while reading the file, the last language requested will be used. (If no language has been requested, the language **usa_english** is assumed.) For a description of the content of files in **/lib/cftime**, see *cftime*(4).

PATH The sequence of directory prefixes that *sh*(1), *time*(1), *nice*(1), *nohup*(1), etc., apply in searching for a file known by an incomplete path name. The prefixes are separated by colons (:). *login*(1) sets **PATH=:/bin:/usr/bin**. [For more detail, see the "Execution" section of the *sh*(1) manual page.]

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

TZ Time zone information. The simplest format is **xxx***n***zzz** where **xxx** is the standard local time zone abbreviation, *n* is the difference in hours from GMT (Greenwich Mean Time), and **zzz** is the abbreviation for an alternate time zone (usually the daylight-saving local time zone), if any; for example,

TZ="EST8EDT"

The most complex format allows you to specify the difference in hours of the alternate time zone from GMT and the starting day and time and ending day and time for using this alternate time zone. For example, in 1985 the complex format corresponding to the above simple example is:

TZ="EST5:00:00EDT4:00:00;118/2:00:00,300/2:00:00"

When the above complex format is used, it must be surrounded by double quotes. For more details, see *ctime*(3C) and *timezone*(4).

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

cat(1), cftime(4), chrtbl(1M), ctime(3C), ctype(3C), date(1), ed(1), env(1), exec(2), login(1), ls(1), mm(1), nice(1), nohup(1), passwd(4), profile(4), sh(1), sort(1) time(1), timezone(4), vi(1).

NOTES

References to the *cftime*(4), *ctime*(3C), and *ctype(3C)* manual pages refer to programming capabilities available beginning with CTIX release 6.1.

Administrators should note the following: if you attempt to set the current date to one of the dates that the standard and alternate time zones change (for example, the date that daylight time is starting or ending), and you attempt to example, the date that daylight time is starting or ending), and you attempt to set the time to a time in the interval between the end of standard time and the beginning of the alternate time (or the end of the alternate time and the beginning of standard time), the results are unpredictable.

NAME

eqnchar - special character definitions for eqn and neqn

SYNOPSIS

eqn /usr/pub/eqnchar [files] | **troff** [options]

neqn /usr/pub/eqnchar [files] | **nroff** [options]

DESCRIPTION

The *eqnchar* contains *troff*(1) and *nroff* character definitions for constructing characters that are not available on the Wang Laboratories, Inc. C/A/T phototypesetter. These definitions are primarily intended for use with *eqn*(1) and *neqn*; *eqnchar* 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*	$\underline{\in}$
andsign	∧	*3dot*	⋮	*subset*	⊂
=del	≜	*thf*	∴	*supset*	⊃
oppA	∀	*quarter*	¼	*!subset*	⊆
oppE	∃	*3quarter*	¾	*!supset*	⊇
angstrom	Å	*degree*	°	*scrL*	ℓ
==<	≦	*==>*	≧		

FILES

/usr/pub/eqnchar

SEE ALSO

eqn(1), nroff(1), troff(1).

NAME

fcntl - file control options

SYNOPSIS

#include <fcntl.h>

DESCRIPTION

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

```
/* Flag values accessible to open(2) and fcntl(2) */
/*  (The first three can only be set by open) */

#define O_RDONLY 0
#define O_WRONLY 1
#define O_RDWR   2
#define O_NDELAY 04       /* Non-blocking I/O */
#define O_APPEND 010      /* append (writes guaranteed at the end) */
#define O_SYNC   020      /* synchronous write option */
#define O_DIRECT 020000/* Perform direct I/O */
#define O_NODIRECT        040000/* NO direct I/O */

/* Flag values accessible only to open(2) */
#define O_CREAT  00400 /* open with file create (uses third open arg)*/
#define O_TRUNC  01000 /* open with truncation */
#define O_EXCL   02000 /* exclusive open */

/* fcntl(2) requests */
#define F_DUPFD  0        /* Duplicate fildes */
#define F_GETFD  1        /* Get fildes flags */
#define F_SETFD  2        /* Set fildes flags */
#define F_GETFL  3        /* Get file flags */
#define F_SETFL  4        /* Set file flags */
#define F_GETLK  5        /* Get file lock */
#define F_SETLK  6        /* Set file lock */
#define F_SETLKW 7        /* Set file lock and wait */
#define F_CHKFL  8        /* Check legality of file flag changes */

/* file segment locking control structure */
struct flock {
        short  l_type;
        short  l_whence;
        long   l_start;
        long   l_len;      /* if 0 then until EOF */
        short  l_pid;      /* returned with F_GETLK*/
```

```
        short l_sysid;     /* returned with F_GETLK*/
    }
    /* file segment locking types */
    #define F_RDLCK   01      /* Read lock */
    #define F_WRLCK   02      /* Write lock */
    #define F_UNLCK   03      /* Remove locks */
```

SEE ALSO

fcntl(2), open(2).

NAME

man - macros for formatting manual pages

SYNOPSIS

nroff -man files

DESCRIPTION

The *man* macros are provided to format *troff*(1) files to look like the entries in this manual.

Any *text* argument listed below can be one to six "words". Double quotation marks ("") can be used to include blanks in a "word." If *text* is empty, the special treatment is applied to the next line that contains text to be printed. For example, **.I** can be used to italicize a whole line, or **.SM** followed by **.B** to make small bold text. By default, hyphenation is enabled for *troff*.

Type font and size are reset to default values before each paragraph and after processing font- and size-setting macros, for example, **.I**, **.RB**, **.SM**. Tab stops are neither used nor set by any macro except **.DT** and **.TH**.

Default units for indents *in* are ens. When *in* is omitted, the previous indent is used. This remembered indent is set to its default value (7.2 ens in *troff*, 5 ens in *nroff*—this corresponds to 0.5´´ in the default page size) by **.TH**, **.P**, and **.RS**, and restored by **.RE**.

.TH *t s c n* Set the title and entry heading: *t* is the title; *s* is the section number; *c* is extra commentary, for example "local;" *n* is new manual name. Invokes **.DT** (see below).

.SH *text* Place subhead *text*, for example, **SYNOPSIS**, here.

.SS *text* Place sub-subhead *text*, for example, **Options**, here.

.B *text* Make *text* bold.

.I *text* Make *text* italic.

.SM *text* Make *text* 1 point smaller than default point size.

.RI *a b* Concatenate roman *a* with italic *b*, and alternate these two fonts for up to six arguments. Similar macros alternate between any two of roman, italic, and bold:

.IR .RB .BR .IB .BI

.P Begin a paragraph with normal font, point size, and indent. **.PP** is a synonym for **.P**.

.HP *in* Begin paragraph with hanging indent.

.TP *in* Begin indented paragraph with hanging tag. The next line that contains text to be printed is taken as the tag. If the tag does not fit, it is printed on a separate line.

.IP *t in*	Same as **.TP** *in* with tag *t*; often used to get an indented paragraph without a tag.
.RS *in*	Increase relative indent (initially zero). Indent all output an extra *in* units from the current left margin.
.RE *k*	Return to the *k*th relative indent level (initially, *k*=1; *k*=0 is equivalent to *k*=1); if *k* is omitted, return to the most recent lower indent level.
.PM *m*	Produces proprietary markings, where *m* can be **P** for **PRIVATE**; **N** for **NOTICE**; **BP** for **BELL LABORATORIES PROPRIETARY**; or **BR** for **BELL LABORATORIES RESTRICTED.**
.DT	Restore default tab settings (every 7.2 ens in *troff*, 5 ens in *nroff*).
.PD *v*	Set the interparagraph distance to *v* vertical spaces. If *v* is omitted, set the interparagraph distance to the default value (0.4v in *troff*, 1v in *nroff*).

The following *strings* are defined:

***R**	® in *troff*, **(Reg.)** in *nroff*.
***S**	Change to default type size.
***(Tm**	Trademark indicator.

The following *number registers* are given default values by **.TH**:

IN	Left margin indent relative to subheads (default is 7.2 ens in *troff*, 5 ens in *nroff*).
LL	Line length including **IN**.
PD	Current interparagraph distance.

FILES

/usr/lib/tmac/tmac.an
/usr/lib/macros/cmp.[nt].[dt].an
/usr/lib/macros/ucmp.[nt].an

SEE ALSO

nroff(1).

CAVEATS

In addition to the macros, strings, and number registers mentioned above, there are defined a number of *internal* macros, strings, and number registers. Except for names predefined by *troff* and number registers **d**, **m**, and **y**, all such internal names are of the form *XA*, where *X* is one of),], and }, and *A* stands for any alphanumeric character.

If a manual entry needs to be preprocessed by *cw*(1), *eqn*(1) [or *neqn*], and/or *tbl*(1), it must begin with a special line [described in *man*(1)], causing the *man* command to invoke the appropriate preprocessor(s).

The programs that prepare the Table of Contents and the Permuted Index for this Manual assume the *NAME* section of each entry consists of a single line of input that has the following format:

name[, name, name ...] \- explanatory text

The macro package increases the interword spaces (to eliminate ambiguity) in the *SYNOPSIS* section of each entry.

The macro package itself uses only the roman font so that one can replace, for example, the bold font by the constant-width font [see *cw*(1)]. Of course, if the input text of an entry contains requests for other fonts (such as **.I**, **.RB**, **\fI**), the corresponding fonts must be mounted.

WARNING

If the argument to **.TH** contains *any* blanks, make sure it is enclosed by double quotation marks (" ").

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 an 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	$\pi/2$.
M_PI_4	$\pi/4$.
M_1_PI	$1/\pi$.
M_2_PI	$2/\pi$.
M_2_SQRTPI	$2/\sqrt{\pi}$.
M_SQRT2	The positive square root of 2.
M_SQRT1_2	The positive square root of 1/2.

For the definitions of various machine-dependent "constants," see the description of the *<values.h>* header file.

FILES

/usr/include/math.h

SEE ALSO

intro(3), matherr(3M), values(5).

NAME

me - macros for formatting papers

SYNOPSIS

nroff -me [options] file ...
troff -me [options] file ...

DESCRIPTION

This package of *nroff* and *troff* macro definitions provides a formatting facility for technical papers in various formats. When producing two-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. These requests can be used after the first .pp, however:

.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.

FILES

/usr/lib/tmac/tmac.e
/usr/lib/me/*

SEE ALSO

eqn(1), troff(1), refer(1), tbl(1).
The -me Reference Manual, Eric P. Allman
Writing Papers with Nroff Using -me

REQUESTS

In the following list, ''initialization'' refers to the first .pp, .lp, .ip, .np, .sh, or .uh macro. This list is incomplete; see *The -me Reference Manual* for a more detailed description.

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.
.++ *m H*	-	no	Define paper section. *m* defines the part of the paper and can be **C** (chapter), **A** (appendix), **P** (preliminary; for example, abstract, table of contents, etc.), **B** (bibliography), **RC** (chapters renumbered from page one each chapter), or **RA** (appendix renumbered from page one).
.+c *T*	-	yes	Begin chapter (or appendix, etc., as set by .++). *T* is the chapter title.
.1c	1	yes	One column format on a new page.
.2c	1	yes	Two column format.
.EN	-	yes	Space after equation produced by *eqn* or *neqn*.
.EQ *x y*	-	yes	Precede equation; break out and add space. Equation number is *y*. The optional argument *x* may be *I* to indent equation (default), *L* to left-adjust the equation, or *C* to center the equation.
.GE	-	yes	End *gremlin* picture.
.GS	-	yes	Begin *gremlin* picture.

.PE	-	yes	End *pic* picture.
.PS	-	yes	Begin *pic* picture.
.TE	-	yes	End table.
.TH	-	yes	End heading section of table.
.TS *x*	-	yes	Begin table; if *x* is *H* table has repeated heading.
.ac *A N*	-	no	Set up for ACM style output. *A* is the Author's name(s), *N* is the total number of pages. Must be given before the first initialization.
.b *x*	no	no	Print x in boldface; if no argument switch to boldface.
.ba +*n*	0	yes	Augments the base indent by *n*. This indent is used to set the indent on regular text (like paragraphs).
.bc	no	yes	Begin new column.
.bi *x*	no	no	Print *x* in bold italics (nofill only).
.bu	-	yes	Begin bulleted paragraph.
.bx *x*	no	no	Print *x* in a box (nofill only).
.ef *´x´y´z´*	´´´´	no	Set even footer to x y z.
.eh *´x´y´z´*	´´´´	no	Set even header to x y z.
.fo *´x´y´z´*	´´´´	no	Set footer to x y z.
.hx	-	no	Suppress headers and footers on next page.
.he *´x´y´z´*	´´´´	no	Set header to x y z.
.hl	-	yes	Draw a horizontal line.
.i *x*	no	no	Italicize *x*; if *x* missing, italic text follows.
.ip *x y*	no	yes	Start indented paragraph, with hanging tag *x*. Indention 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.
.sm *x*	-	no	Set *x* in a smaller pointsize.
.sz +*n*	10p	no	Augment the point size by *n* points.
.th	no	no	Produce the paper in thesis format. Must be given before initialization.
.tp	no	yes	Begin title page.
.u *x*	-	no	Underline argument (even in *troff*). (Nofill only).
.uh	-	yes	Like .sh but unnumbered.
.xp *x*	-	no	Print index *x*.

NAME

mm - the MM macro package for formatting documents

SYNOPSIS

mm [options] [files]

nroff -mm [options] [files]

nroff -cm [options] [files]

DESCRIPTION

This package provides a formatting capability for a very wide variety of documents. The manner in which a document is typed in and edited is essentially independent of whether the document is to be eventually formatted at a terminal or is to be phototypeset. See the references below for further details.

The **-mm** option causes *nroff* and *troff*(1) to use the non-compacted version of the macro package, while the **-cm** option results in the use of the compacted version, thus speeding up the process of loading the macro package.

FILES

/usr/lib/tmac/tmac.	pointer to the non-compacted version of the package
/usr/lib/macros/mm[nt]	non-compacted version of the package
/usr/lib/macros/cmp.[nt].[dt].m	compacted version of the package
/usr/lib/macros/ucmp.[nt].m	initializers for the compacted version of the package

SEE ALSO

mm(1), mmt(1), nroff(1).
Programmer's Guide: CTIX Supplement.

NAME

mptx - the macro package for formatting a permuted index

SYNOPSIS

nroff -mptx [options] [files] [options] [files]

DESCRIPTION

This package provides a definition for the **.xx** macro used for formatting a permuted index as produced by *ptx*(1). This package does not provide any other formatting capabilities such as headers and footers. If these or other capabilities are required, the *mptx* macro package may be used in conjuction with the *MM* macro package. In this case, the **-mptx** option must be invoked *after* the **-mm** call. For example:

nroff -cm -mptx file

or

mm -mptx file

FILES

/usr/lib/tmac/tmac.ptx	pointer to the non-compacted version of the package
/usr/lib/macros/ptx	non-compacted version of the package

SEE ALSO

mm(1), nroff(1), ptx(1), mm(5).

NAME

ms - text formatting macros

SYNOPSIS

nroff -ms [options] file ...
troff -ms [options] file ...

DESCRIPTION

This package of *nroff* and *troff* 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*. 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** italicizes *word*. Output of the *tbl* and *eqn* preprocessors for equations and tables is acceptable as input.

FILES

/usr/lib/tmac/tmac.x
/usr/lib/ms/x.???

SEE ALSO

eqn(1), tbl(1), troff(1).

REQUESTS

Macro Name	Initial Value	Break? Reset?	Explanation
.AB *x*	-	y	begin abstract; if *x*=no don't 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
.BT	date	n	bottom title, printed at foot of page

.BX *x*	-	n	print word *x* in a box
.CM	if t	n	cut mark between pages
.CT	-	y,y	chapter title: page number moved to CF (TM only)
.DA *x*	if n	n	force date *x* at bottom of page; today if no *x*
.DE	-	y	end display (unfilled text) of any kind
.DS *x y*	I	y	begin display with keep; *x*=I,L,C,B; *y*=indent
.ID *y*	8n,.5i	y	indented display with no keep; *y*=indent
.LD	-	y	left display with no keep
.CD	-	y	centered display with no keep
.BD	-	y	block display; center entire block
.EF *x*	-	n	even page footer *x* (3 part as for .tl)
.EH *x*	-	n	even page header *x* (3 part as for .tl)
.EN	-	y	end displayed equation produced by *eqn*
.EQ *x y*	-	y	break out equation; *x*=L,I,C; *y*=equation number
.FE	-	n	end footnote to be placed at bottom of page
.FP	-	n	numbered footnote paragraph; may be redefined
.FS *x*	-	n	start footnote; *x* is optional footnote label
.HD	undef	n	optional page header below header margin
.I *x*	-	n	italicize *x*; if no *x*, switch to italics
.IP *x y*	-	y,y	indented paragraph, with hanging tag *x*; *y*=indent
.IX *x y*	-	y	index words *x y* and so on (up to 5 levels)
.KE	-	n	end keep of any kind
.KF	-	n	begin floating keep; text fills remainder of page
.KS	-	y	begin keep; unit kept together on a single page
.LG	-	n	larger; increase point size by 2
.LP	-	y,y	left (block) paragraph.
.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 1st 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 1st 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 tabs 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	UC Berkeley thesis mode
.TS *x*	-	y,y	begin table; if *x* =H table has multi-page header
.UL *x*	-	n	underline *x* , even in *troff*
.UX *x*	-	n	UNIX; trademark message first time; *x* appended
.XA *x y*	-	y	another index entry; *x* =page or no for none; *y* =indent
.XE	-	y	end index entry (or series of .IX entries)
.XP	-	y,y	paragraph with first line exdented, others indented
.XS *x y*	-	y	begin index entry; *x* =page or no for none; *y* =indent
.1C	on	y,y	one column format, on a new page
.2C	-	y,y	begin two column format
.[0	-	n	end of unclassifiable type of reference
.[N	-	n	N= 1:journal-article, 2:book, 3:book-article, 4:report

REGISTERS

Formatting distances can be controlled in -ms by means of built-in number registers. For example, this sets the line length to 6.5 inches:

.nr LL 6.5i

Here is a table of number registers and their default values:

Name	**Register Controls**	**Takes Effect**	**Default**
PS	point size	paragraph	10
VS	vertical spacing	paragraph	12
LL	line length	paragraph	6i
LT	title length	next page	same as LL
FL	footnote length	next .FS	5.5i
PD	paragraph distance	paragraph	1v (if n), .3v (if t)
DD	display distance	displays	1v (if n), .5v (if t)
PI	paragraph indent	paragraph	5n
QI	quote indent	next .QP	5n
FI	footnote indent	next .FS	2n
PO	page offset	next page	0 (if n), ~1i (if t)
HM	header margin	next page	1i
FM	footer margin	next page	1i
FF	footnote format	next .FS	0 (1, 2, 3 available)

When resetting these values, make sure to specify the appropriate units. Setting the line length to 7, for example, results 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 a .IP-like footnote paragraph.

Here is a list of string registers available in -ms; they can 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.

BUGS

Floating keeps and regular keeps are diverted to the same space, so they cannot be mixed together with predictable results.

NAME

mv - a troff macro package for typesetting view graphs and slides

SYNOPSIS

mvt [**-a**] [options] [files]

troff [**-a**] [**-rX1**] **-mv** [options] [files]

DESCRIPTION

This package makes it easy to typeset view graphs and projection slides in a variety of sizes. A few macros (briefly described below) accomplish most of the formatting tasks needed in making transparencies. All of the facilities of *troff*(1), *cw*(1), *eqn*(1), and *tbl*(1) are available for more difficult tasks.

The output can be previewed on most terminals, and, in particular, on the Tektronix 4014, as well as on the Versatec printer. For these two devices, specify the **-rX1** option (this option is automatically specified by the *mvt* command-q.v.-when that command is invoked with the **-T4014** or **-Tvp** options). To preview output on other terminals, specify the **-a** option.

The available macros are:

.VS [*n*] [*i*] [*d*] Foil-start macro; foil size is to be 7″×7″; *n* is the foil number, *i* is the foil identification, *d* is the date; the foil-start macro resets all parameters (indent, point size, etc.) to initial default values, except for the values of *i* and *d* arguments inherited from a previous foil-start macro; it also invokes the .A macro (see below).

The naming convention for this and the following eight macros is that the first character of the name (V or S) distinguishes between view graphs and slides, respectively, while the second character indicates whether the foil is square (S), small wide (w), small high (h), big wide (W), or big high (H). Slides are "skinnier" than the corresponding view graphs: the ratio of the longer dimension to the shorter one is larger for slides than for view graphs. As a result, slide foils can be used for view graphs, but not vice versa; on the other hand, view graphs can accommodate a bit more text.

.Vw [*n*] [*i*] [*d*] Same as .VS, except that foil size is 7″ wide × 5″ high.
.Vh [*n*] [*i*] [*d*] Same as .VS, except that foil size is 5″×7″.
.VW [*n*] [*i*] [*d*] Same as .VS, except that foil size is 7″×5.4″.
.VH [*n*] [*i*] [*d*] Same as .VS, except that foil size is 7″×9″.

.Sw [*n*] [*i*] [*d*]	Same as **.VS**, except that foil size is 7″×5″.
.Sh [*n*] [*i*] [*d*]	Same as **.VS**, except that foil size is 5″×7″.
.SW [*n*] [*i*] [*d*]	Same as **.VS**, except that foil size is 7″×5.4″.
.SH [*n*] [*i*] [*d*]	Same as **.VS**, except that foil size is 7″×9″.
.A [*x*]	Place text that follows at the first indentation level (left margin); the presence of *x* suppresses the ½ line spacing from the preceding text.
.B [*m* [*s*]]	Place text that follows at the second indentation level; text is preceded by a mark; *m* is the mark (default is a large bullet); *s* is the increment or decrement to the point size of the mark with respect to the *prevailing* point size (default is 0); if *s* is 100, it causes the point size of the mark to be the same as that of the *default* mark.
.C [*m* [*s*]]	Same as **.B**, but for the third indentation level; default mark is a dash.
.D [*m* [*s*]]	Same as **.B**, but for the fourth indentation level; default mark is a small bullet.
.T *string*	*String* is printed as an over-size, centered title.
.I [*in*] [*a* [*x*]]	Change the current text indent (does not affect titles); *in* is the indent (in inches unless dimensioned, default is 0); if *in* is signed, it is an increment or decrement; the presence of *a* invokes the **.A** macro (see below) and passes *x* (if any) to it.
.S [*p*] [*l*]	Set the point size and line length; *p* is the point size (default is "previous"); if *p* is 100, the point size reverts to the *initial* default for the current foil-start macro; if *p* is signed, it is an increment or decrement (default is 18 for **.VS**, **.VH**, and **.SH**, and 14 for the other foil-start macros); *l* is the line length (in inches unless dimensioned; default is 4.2″ for **.Vh**, 3.8″ for **.Sh**, 5″ for **.SH**, and 6″ for the other foil-start macros).
.DF *n f* [*n f*...]	Define font positions; may not appear within a foil's input text (i.e., it may only appear after all the input text for a foil, but before the next foil-start macro); *n* is the position of font *f*; up to four "*n f*" pairs may be specified; the first font named becomes the *prevailing* font; the initial setting is (**H** is a synonym for **G**): .DF 1 H 2 I 3 B 4 S

.DV [*a*] [*b*] [*c*] [*d*]	Alter the vertical spacing between indentation levels; *a* is the spacing for **.A**, *b* is for **.B**, *c* is for **.C**, and *d* is for **.D**; all non-null arguments must be dimensioned; null arguments leave the corresponding spacing unaffected; initial setting is: .DV .5v .5v .5v 0v
.U *str1* [*str2*]	Underline *str1* and concatenate *str2* (if any) to it.

The last four macros in the above list do not cause a break; the **.I** macro causes a break only if it is invoked with more than one argument; all the other macros cause a break.

The macro package also recognizes the following upper-case synonyms for the corresponding lower-case *troff* requests:

.AD .BR .CE .FI .HY .NA .NF .NH .NX .SO .SP .TA .TI

The **Tm** string produces the trademark symbol.

The input tilde (~) character is translated into a blank on output.

See the user's manual cited below for further details.

FILES

/usr/lib/tmac/tmac.v
/usr/lib/macros/vmca

SEE ALSO

cw(1), eqn(1), mmt(1), tbl(1), troff(1).
A Macro Package for View Graphs and Slides by T. A. Dolotta and D. W. Smith.

BUGS

The .VW and .SW foils are meant to be 9″ wide by 7″ high, but because the typesetter paper is generally only 8″ wide, they are printed 7″ wide by 5.4″ high and have to be enlarged by a factor of 9/7 before use as view graphs; this makes them less than totally useful.

NAME

prof - profile within a function

SYNOPSIS

#define MARK
#include <prof.h>

void MARK (name)

DESCRIPTION

MARK will introduce a mark called *name* that will be treated the same as a function entry point. Execution of the mark will add to a counter for that mark, and program-counter time spent will be 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 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. This may be defined by a preprocessor directive as in the synopsis, or by a command line argument, i.e:

```
cc -p -DMARK foo.c
```

If MARK is not defined, the *MARK*(name) statements may be left in the source files containing them and will be ignored.

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

prof(1), profil(2), monitor(3C).

NAME

regexp - regular expression compile and match routines

SYNOPSIS

```
#define INIT <declarations>
#define GETC() <getc code>
#define PEEKC() <peekc code>
#define UNGETC(c) <ungetc code>
#define RETURN(pointer) <return code>
#define ERROR(val) <error code>

#include <regexp.h>

char *compile (instring, expbuf, endbuf, eof)
char *instring, *expbuf, *endbuf;
int eof;

int step (string, expbuf)
char *string, *expbuf;

extern char *loc1, *loc2, *locs;

extern int circf, sed, nbra;
```

DESCRIPTION

This page describes general-purpose regular expression matching routines in the form of *ed*(1), defined in **<regexp.h>**. Programs such as *ed*(1), *sed*(1), *grep*(1), *bs*(1), *expr*(1), etc., which perform regular expression matching use this source file. In this way, only this file need be changed to maintain regular expression compatibility.

The interface to this file is unpleasantly complex. Programs that include this file must have the following five macros declared before the ''#include <regexp.h>'' statement. These macros are used by the *compile* routine.

GETC() — Return the value of the next character in the regular expression pattern. Successive calls to GETC() should return successive characters of the regular expression.

PEEKC() — Return the next character in the regular expression. Successive calls to PEEKC() should return the same character [which should also be the next character returned by GETC()].

UNGETC(*c*) — Cause 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 value of the macro UNGETC(*c*) is always ignored.

RETURN(*pointer*)
This macro is used on normal exit of the *compile* routine. The value of the argument *pointer* is a pointer to the character after the last character of the compiled regular expression. This is useful to programs that have memory allocation to manage.

ERROR(*val*)
This is the abnormal return from the *compile* routine. The argument *val* is an error number (see table below for meanings). This call should never return.

ERROR	MEANING
11	Range endpoint too large.
16	Bad number.
25	``\digit'' out of range.
36	Illegal or missing delimiter.
41	No remembered search string.
42	\(\) imbalance.
43	Too many \(.
44	More than 2 numbers given in \{ \}.
45	} expected after \.
46	First number exceeds second in \{ \}.
49	[ ] imbalance.
50	Regular expression overflow.

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 that 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. For example, in *ed*(1), this character is usually a /.

Each program that includes this file must have a **#define** statement for INIT. This definition will be placed right after the declaration for the function *compile* and the opening curly brace ({). It is used for dependent declarations and initializations. Most often it is used to set a register variable to point 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 the example below of the declarations taken from *grep*(1).

There are other functions in this file which perform actual regular expression matching, one of which is the function *step*. The call to *step* is as follows:

> **step(***string***,expbuf)**

The first parameter to *step* 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 of the function *compile*.

The function *step* returns non-zero if the given string matches the regular expression, and zero if the expressions do not match. If there is a match, two external character pointers are set as a side effect to the call to *step*. The variable set in *step* is *loc1*. This is a pointer to the first character that matched the regular expression. The variable *loc2*, which is set by the function *advance*, points to the character after the last character that matches the regular expression. Thus if the regular expression matches the entire line, *loc1* will point to the first character of *string* and *loc2* will point to the null at the end of *string*.

Step uses the external variable *circf* which is set by *compile* if the regular expression begins with ^. If this is set then *step* will try to match the regular expression to the beginning of the string only. If more than one regular expression is to be compiled before the first is executed the value of *circf* should be saved for each compiled expression and *circf* should be set to that saved value before each call to *step*.

The function *advance* is called from *step* with the same arguments as *step*. The purpose of *step* is to step through the *string* argument and call *advance* until *advance* returns non-zero indicating a match or until the end of *string* is reached. If one wants to constrain *string* to the beginning of the line in all cases, *step* need not be called; simply call *advance*.

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 some time during the backing-up process, *advance* will break out of the loop that backs up and will return zero. This is used by *ed*(1) and *sed*(1) for substitutions done globally (not just the first occurrence, but the whole line) so, for example, expressions like s/y*//g do not loop forever.

The additional external variables *sed* and *nbra* are used for special purposes.

EXAMPLES

The following is an example of how the regular expression macros and calls look from *grep*(1):

```
#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();
```

FILES

/usr/include/regexp.h

SEE ALSO

ed(1), expr(1), grep(1), sed(1).

NAME

stat - data returned by stat system call

SYNOPSIS

#include <sys/types.h>
#include <sys/stat.h>

DESCRIPTION

The system calls *stat* and *fstat* return data whose structure is defined by this include file. The encoding of the field *st_mode* is defined in this file also.

```
/* Structure of the result of stat */

struct stat
{
        dev_t  st_dev;
        ushort st_ino;
        ushort st_mode;
        short  st_nlink;
        ushort st_uid;
        ushort st_gid;
        dev_t  st_rdev;
        off_t  st_size;
        time_t st_atime;
        time_t st_mtime;
        time_t st_ctime;
};

#define S_IFMT    0170000 /* type of file */
#define S_IFDIR   0040000 /* directory */
#define S_IFCHR   0020000 /* character special */
#define S_IFBLK   0060000 /* block special */
#define S_IFREG   0100000 /* regular */
#define S_IFIFO   0010000 /* fifo */
#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

term - conventional names for terminals

DESCRIPTION

These names are used by certain commands [for example, *man*(1), *tabs*(1), *tput*(1), *vi*(1) and *curses*(3X)] and are maintained as part of the shell environment in the environment variable **TERM** [see *sh*(1), *profile*(4), and *environ*(5)].

Entries in *terminfo*(4) source files consist of a number of comma-separated fields. [To get the source description for a terminal, use the **-I** option of *infocmp*(1M).] White space after each comma is ignored. The first line of each terminal description in the *terminfo*(4) database gives the names by which *terminfo*(4) 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 eight characters, chosen from [a-z0-9], 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 would be **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**.

Some of the known terminal names may include the following (for a complete list, type: **ls -C /usr/lib/terminfo/?**):

pt	Convergent Technologies Programmable Terminal
gt	Convergent Technologies Graphics Terminal
ct300	Convergent Technologies TO-300 (Link) Terminal
[hp]2621	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
[hp]2640	Hewlett-Packard 2640 series
[hp]2645	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 terminal 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 terminal in 80-column mode - version 1
4410v1-w,5410v1-w	AT&T 4410/5410 terminal in 132-column mode - version 1
4415,5420	AT&T 4415/5420 in 80-column mode
4415-nl,5420-nl	AT&T 4415/5420 without changing labels
4415-rv,5420-rv	AT&T 4415/5420 80 columns in reverse video
4415-rv-nl,5420-rv-nl	AT&T 4415/5420 reverse video without changing labels
4415-w,5420-w	AT&T 4415/5420 in 132-column mode
4415-w-nl,5420-w-nl	AT&T 4415/5420 in 132-column mode without changing labels
4415-w-rv,5420-w-rv	AT&T 4415/5420 132 columns in reverse video
4415-w-rv-nl	AT&T 4415/5420 132 columns reverse video without changing labels
5420-w-rv-nl	AT&T 5420 132 columns reverse video without changing labels
4418,5418	AT&T 5418 in 80-column mode
4418-w,5418-w	AT&T 5418 in 132-column mode
4420	AT&T Teletype Model 4420
4424	AT&T Teletype Model 4424
4424-2	AT&T Teletype Model 4424 in display function group ii
4425,5425	AT&T 4425/5425
4425-fk,5425-fk	AT&T 4425/5425 without function keys
4425-nl,5425-nl	AT&T 4425/5425 without changing labels in 80-column mode

4425-w,5425-w	AT&T 4425/5425 in 132-column mode
4425-w-fk,5425-w-fk	AT&T 4425/5425 without function keys in 132-column mode
4425-nl-w,5425-nl-w	AT&T 4425/5425 without changing labels in 132-column mode
4426	AT&T Teletype Model 4426S
450	DASI 450 (same as Diablo 1620)
450-12	DASI 450 in 12-pitch mode
500,att500	AT&T-IS 500 terminal
510,510a	AT&T 510/510a in 80-column mode
513bct,att513	AT&T 513 bct terminal
5320	AT&T 5320 hardcopy terminal
5420_2	AT&T 5420 model 2 in 80-column mode
5420_2-w	AT&T 5420 model 2 in 132-column mode
5620,dmd	AT&T 5620 terminal 88 columns
5620-24,dmd-24	AT&T Teletype Model DMD 5620 in a 24x80 layer
5620-34,dmd-34	AT&T Teletype Model DMD 5620 in a 34x80 layer
610,610bct	AT&T 610 bct terminal in 80-column mode
610-w,610bct-w	AT&T 610 bct terminal in 132-column mode
[pc]7300,unix_pc	AT&T UNIX PC Model 7300
735,ti	Texas Instruments TI735 and TI725
745	Texas Instruments TI745
dumb	generic name for terminals that lack reverse line-feed and other special escape sequences
hp	Hewlett-Packard (same as 2645)
lp	generic name for a line printer
pt505	AT&T Personal Terminal 505 (22 lines)
pt505-24	AT&T Personal Terminal 505 (24-line mode)
sync	generic name for synchronous Teletype Model 4540-compatible terminals

Commands whose behavior depends on the type of terminal should accept arguments of the form **-T***term* where *term* is one of the names given above; if no such argument is present, such commands should obtain the terminal type from the environment variable **TERM**, which, in turn, should contain *term*.

FILES

/usr/lib/terminfo/?/* compiled terminal description database

SEE ALSO

man(1), sh(1), stty(1), tabs(1), tput(1), tplot(1G), vi(1) infocmp(1M), curses(3X), profile(4), terminfo(4), environ(5).
UNIX System V Release 3.2 Programmer's Guide.

NOTES

Not all programs follow the above naming conventions.

NAME

types - primitive system data types

SYNOPSIS

#include <sys/types.h>

DESCRIPTION

The data types defined in the include file are used in UNIX system code; some data of these types are accessible to user code:

```
typedef  struct { int r[1]; } *physadr;
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  ushort             ino_t;
typedef  short              cnt_t;
typedef  long               time_t;
typedef  int                label_t[10];
typedef  short              dev_t;
typedef  long               off_t;
typedef  long               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;
```

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 GMT, January 1, 1970. The major and minor parts of a device code specify kind and unit number of a device. Offsets are measured in bytes from the beginning of a file. The *label_t* variables are used to save the processor state while another process is running.

SEE ALSO

fs(4).

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 (e.g., int).

HIBITS The value of a short integer with only the high-order bit set (in most implementations, 0x8000).

HIBITL The value of a long integer with only the high-order bit set (in most implementations, 0x80000000).

HIBITI The value of a regular integer with only the high-order bit set (usually the same as HIBITS or HIBITL).

MAXSHORT The maximum value of a signed short integer (in most implementations, 0x7FFF ≡ 32767).

MAXLONG The maximum value of a signed long integer (in most implementations, 0x7FFFFFFF ≡ 2147483647).

MAXINT The maximum value of a signed regular integer (usually the same as MAXSHORT or MAXLONG).

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.

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.

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.

FILES

/usr/include/values.h

SEE ALSO

intro(3), math(5).

NAME

varargs - handle variable argument list

SYNOPSIS

#include <varargs.h>

va_alist

va_dcl

void va_start(pvar)
va_list pvar;

type **va_arg(pvar,** *type***)**
va_list pvar;

void va_end(pvar)
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 nonportable, 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 returns 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* ... *va_end*, are possible.

EXAMPLE

The following example shows a possible implementation of *execl* [see *exec*(2)].

```
#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];
        int argno = 0;

        va_start(ap);
        file = va_arg(ap, char *);
        while ((args[argno++] = va_arg(ap, char *)) !=
                (char *)0);
        va_end(ap);
        return execv(file, args);
}
```

SEE ALSO

exec(2), printf(3S), vprintf(3S).

NOTES

It is up to the calling routine to specify how many arguments there are, since it is not always possible to determine this 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.

NAME

intro - introduction to games

DESCRIPTION

This section describes the recreational and educational programs found in the directory **/usr/games**. The availability of these programs may vary from system to system.

NAME

advent - explore Colossal Cave

SYNOPSIS

/usr/games/advent

DESCRIPTION

The *advent* game is Adventure, the original computer-moderated role-playing game. It accepts commands of one or two English words and responds by describing situations and how your commands affect them. The object of the game is to retrieve the treasures from Colossal Cave, placing them in the Well House.

Part of the game is figuring out the useful commands, but the following are worth knowing in advance:

help Basic hints.

quit End the game and give final score.

suspend Save the game's current state in a file called **$HOME/adv.susp**. The next time you play the game, you automatically start from where you left off instead of from the beginning.

FILES

/usr/games/advfiles/*
$HOME/adv.susp

WARNINGS

Kibitzing this sort of game properly is a fine art. People who tell you about the shortcuts can spoil the game, especially in the early stages.

Some movement verbs, such as **follow**, work only well enough to get you lost. Compass points are more (but not completely) reliable.

Only the first five characters of an input word are significant.

The command vocabulary and control of objects is limited. But discovering limitations has become part of the game.

NAME

arithmetic - provide drill in number facts

SYNOPSIS

/usr/games/arithmetic [+-x/] [range]

DESCRIPTION

The *arithmetic* game types out simple arithmetic problems, and waits for an answer to be typed in. If the answer is correct, it types back ''Right!'', and a new problem. If the answer is wrong, it replies ''What?'', and waits for another answer. Every twenty problems, it publishes statistics on correctness and the time required to answer.

To quit the program, type an interrupt (delete).

The first optional argument determines the kind of problem to be generated; +, -, x, and / respectively cause addition, subtraction, multiplication, and division problems to be generated. One or more characters can be given; if more than one is given, the different types of problems are mixed in random order; the default is +-.

range is a decimal number; all addends, subtrahends, differences, multiplicands, divisors, and quotients are less than or equal to the value of *range*. Default *range* is 10.

At the start, all numbers less than or equal to *range* are equally likely to appear. If the respondent makes a mistake, the numbers in the missed problem become more likely to reappear.

As a matter of educational philosophy, the program does not give correct answers since the learner should, in principle, be able to calculate them. Thus the program is intended to provide drill for someone just past the first learning stage, not to teach number facts *de novo*. For almost all users, the relevant statistic should be time per problem, not percent correct.

NAME

back - the game of backgammon

SYNOPSIS

/usr/games/back

DESCRIPTION

The *back* game is a program that provides a partner for the game of backgammon. It is designed to play at three different levels of skill, one of which you must select. In addition to selecting the opponent's level, you may also indicate that you would like to roll your own dice during your turns (for the superstitious players). You are also given the opportunity to move first. The practice of each player rolling one die for the first move is not incorporated.

The points are numbered 1-24, with 1 being white's extreme inner table, 24 being brown's inner table, 0 being the bar for removed white pieces and 25 the bar for brown. For details on how moves are expressed, type **y** when *back* asks, "Instructions?" at the beginning of the game. When *back* first asks, "Move?", type **?** to see a list of move options other than entering your numerical move.

When the game is finished, *back* asks if you want the log. If you respond with **y**, *back* attempts to append to or create a file **back.log** in the current directory.

FILES

/usr/games/lib/backrules	rules file
/tmp/b*	log temp file
back.log	log file

WARNINGS

The only level really worth playing is "expert," and it plays only the forward game.

The *back* game complains loudly if you attempt to make too *many* moves in a turn, but it becomes very silent if you make too *few*.

BUGS

Doubling is not implemented.

The *back* game occasionally disallows a legal move when you have a man on the bar.

NAME

bj - the game of black jack

SYNOPSIS

/usr/games/bj

DESCRIPTION

The *bj* game is a serious attempt at simulating the dealer in the game of black jack (or twenty-one) as might be found in Reno. The following rules apply:

The bet is $2 every hand.

> A player "natural" (black jack) pays $3. A dealer natural loses $2. Both dealer and player naturals is a "push" (no money exchange).
>
> If the dealer has an ace up, the player is allowed to make an "insurance" bet against the chance of a dealer natural. If this bet is not taken, play resumes as normal. If the bet is taken, it is a side bet where the player wins $2 if the dealer has a natural and loses $1 if the dealer does not.
>
> If the player is dealt two cards of the same value, he is allowed to "double". He is allowed to play two hands, each with one of these cards. (The bet is doubled also; $2 on each hand.)
>
> If a dealt hand has a total of ten or eleven, the player may "double down". He may double the bet ($2 to $4) and receive exactly one more card on that hand.
>
> Under normal play, the player may "hit" (draw a card) as long as his total is not over twenty-one. If the player "busts" (goes over twenty-one), the dealer wins the bet.
>
> When the player "stands" (decides not to hit), the dealer hits until he attains a total of seventeen or more. If the dealer busts, the player wins the bet.
>
> If both player and dealer stand, the one with the largest total wins. A tie is a push.

The machine deals and keeps score. The following prompts appear appropriate times. Each question is answered by **y** followed by a new-line for "yes", or just new-line for "no".

```
?               (means, "do you want a hit?")
Insurance?
Double down?
```

Every time the deck is shuffled, the dealer so states and the "action" (total bet) and "standing" (total won or lost) is printed. To exit, press the interrupt key (DEL) and the action and standing is printed.

NAME

craps - the game of craps

SYNOPSIS

/usr/games/craps

DESCRIPTION

The *craps* game is a form of the game of craps that is played in Las Vegas. The program simulates the *roller*, while the user (the *player*) places bets. The player may choose, at any time, to bet with the roller or with the *House*. A bet of a negative amount is taken as a bet with the House, any other bet is a bet with the roller.

The player starts off with a ''bankroll'' of $2,000.

The program prompts with:

bet?

The bet can be all or part of the player's bankroll. Any bet over the total bankroll is rejected and the program prompts with **bet?** until a proper bet is made.

Once the bet is accepted, the roller throws the dice. The following rules apply (the player wins or loses depending on whether the bet is placed with the roller or with the House; the odds are even). The *first* roll is the roll immediately following a bet:

1. On the first roll:

7 or 11	wins for the roller;
2, 3, or 12	wins for the House;
any other number	is the *point*, roll again (Rule 2 applies).

2. On subsequent rolls:

point	roller wins;
7	House wins;
any other number	roll again.

If a player loses the entire bankroll, the House offers to lend the player an additional $2,000. The program prompts:

marker?

A yes (or y) consummates the loan. Any other reply terminates the game.

If a player owes the House money, the House reminds the player, before a bet is placed, how many markers are outstanding.

If, at any time, the bankroll of a player who has outstanding markers exceeds $2,000, the House asks:

Repay marker?

A reply of **yes** (or **y**) indicates the player's willingness to repay the loan. If only 1 marker is outstanding, it is immediately repaid. However, if more than 1 marker are outstanding, the House asks:

How many?

markers the player would like to repay. If an invalid number is entered (or just a carriage return), an appropriate message is printed and the program prompts with **How many?** until a valid number is entered.

If a player accumulates 10 markers (a total of $20,000 borrowed from the House), the program informs the player of the situation and exits.

Should the bankroll of a player who has outstanding markers exceed $50,000, the *total* amount of money borrowed is *automatically* repaid to the House.

Any player who accumulates $100,000 or more breaks the bank. The program then prompts:

New game?

to give the House a chance to win back its money.

Any reply other than **yes** is considered to be a **no** (except in the case of **bet?** or **How many?**). To exit, send an interrupt (break), DEL, or control-D. The program indicates whether the player won, lost, or broke even.

MISCELLANEOUS

The random number generator for the die numbers uses the seconds from the time of day. Depending on system usage, these numbers, at times, may seem strange but occurrences of this type in a real dice situation are not uncommon.

NAME

fish - play ''Go Fish''

SYNOPSIS

/usr/games/fish

DESCRIPTION

The *fish* game plays the game of ''Go Fish'', a children's card game. The Object is to accumulate 'books' of 4 cards with the same face value. The players alternate turns; each turn begins with one player selecting a card from his hand, and asking the other player for all cards of that face value. If the other player has one or more cards of that face value in his hand, he gives them to the first player, and the first player makes another request. Eventually, the first player asks for a card that is not in the second player's hand: he replies 'GO FISH!' The first player then draws a card from the 'pool' of undealt cards. If this is the card he had last requested, he draws again. When a book is made, either through drawing or requesting, the cards are laid down and no further action takes place with that face value.

To play the computer, simply make guesses by typing one of the following when asked: 2, 3, 4, 5, 6, 7, 8, 9, 10, j, q, k, or a. Pressing return gives you information about the size of my hand and the pool, and tells you about my books. Saying 'p' as a first guess puts you into 'pro' level; the default is pretty dumb.

NAME

fortune - print a random, hopefully interesting, adage

SYNOPSIS

/usr/games/fortune [**-**] [**-wslao**]

DESCRIPTION

The *fortune* command with no arguments prints out a random adage. The flags mean:

-w Waits before termination for an amount of time calculated from the number of characters in the message. This is useful if it is executed as part of the logout procedure to guarantee that the message can be read before the screen is cleared.

-s Short messages only.

-l Long messages only.

-o Choose from an alternate list of adages, often used for potentially offensive ones.

-a Choose from either list of adages.

FILES

/usr/games/lib/fortunes.dat

NAME

hangman - guess the word

SYNOPSIS

/usr/games/hangman [arg]

DESCRIPTION

The *hangman* game chooses a word at least seven letters long from a dictionary. The user is to guess letters one at a time.

The optional argument *arg* names an alternate dictionary.

FILES

/usr/lib/w2006

BUGS

Hyphenated compounds are run together.

NAME

maze - generate a maze

SYNOPSIS

/usr/games/maze [**seed** [**d**] [**n**] [**b**]]

DESCRIPTION

The *maze* game prints a maze. It uses the system clock as the random number seed. If **seed** is specified, *maze* uses it as the seed and shows the solution. An **n** suppresses the solution, a **b** shows backouts, and a **d** provides debugging information.

BUGS

Some mazes (especially small ones) have no solutions.

NAME

moo - guessing game

SYNOPSIS

/usr/games/moo

DESCRIPTION

The *moo* game is a guessing game imported from England. The computer picks a number consisting of four distinct decimal digits. The player guesses four distinct digits being scored on each guess. A ''cow'' is a correct digit in an incorrect position. A ''bull'' is a correct digit in a correct position. The game continues until the player guesses the number (a score of four bulls).

NAME

number - convert Arabic numerals to English

SYNOPSIS

/usr/games/number

DESCRIPTION

The *number* game copies the standard input to the standard output, changing each decimal number to a fully spelled out version.

NAME

quiz - test your knowledge

SYNOPSIS

/usr/games/quiz [**-i** file] [**-t**] [category1 category2]

DESCRIPTION

The *quiz* game gives associative knowledge tests on various subjects. It asks items chosen from *category1* and expects answers from *category2*, or vice versa. If no categories are specified, *quiz* gives instructions and lists the available categories.

The *quiz* game tells a correct answer whenever you type a bare new-line. At the end of input, upon interrupt, or when questions run out, *quiz* reports a score and terminates.

The **-t** flag specifies "tutorial" mode, where missed questions are repeated later, and material is gradually introduced as you learn.

The -i flag causes the named file to be substituted for the default index file. The lines of these files have the syntax:

```
line      = category new-line | category : line
category  = alternate | category | alternate
alternate = empty | alternate primary
primary   = character | [ category ] | option
option    = { category }
```

The first category on each line of an index file names an information file. The remaining categories specify the order and contents of the data in each line of the information file. Information files have the same syntax. Backslash (\) is used as with *sh*(1) to quote syntactically significant characters or to insert transparent new-lines into a line. When either a question or its answer is empty, *quiz* refrains from asking it.

FILES

/usr/games/lib/quiz/index
/usr/games/lib/quiz/*

BUGS

The construct "a|ab" does not work in an information file. Use "a{b}".

NAME

trk - trekkie game

SYNOPSIS

/usr/games/trk [[**-a**] file]

DESCRIPTION

The *trk* game is a game of space glory and war. Below is a summary of commands. For complete documentation, see *Trek* by Eric Allman.

If a filename is given, a log of the game is written onto that file. If the **-a** flag is given before the filename, that file is appended to, not truncated.

The game asks you what length game you would like. Valid responses are "short", "medium", and "long". You may also type "restart", which restarts a previously saved game. You are then prompted for the skill, to which you must respond "novice", "fair", "good", "expert", "commadore", or "impossible". You should normally start out as a novice and work up.

In general, throughout the game, if you forget what is appropriate, the game tells you what it expects if you just type in a question mark.

COMMAND SUMMARY

abandon	**phasers manual** amt1 course1 spread1 ...
capture	**torpedo** course [**yes**] angle/**no**
cloak up/down	**ram** course distance
computer request; ...	**rest** time
damages	**shell**
destruct	**shields up/down**
dock	**srscan** [**yes/no**]
help	**status**
impulse course distance	**terminate** [**yes/no**]
lrscan	**undock**
move course distance	**visual** course
phasers automatic amount	**warp** warp_factor

NAME

ttt, cubic - tic-tac-toe

SYNOPSIS

/usr/games/ttt
/usr/games/cubic

DESCRIPTION

The *ttt* game is the X and O game popular in the first grade. This is a learning program that never makes the same mistake twice.

Although it learns, it learns slowly. It must lose nearly 80 games to completely know the game.

cubic plays three-dimensional tic-tac-toe on a 4×4×4 board. Moves are specified as a sequence of three coordinate numbers in the range 1-4.

FILES

/usr/games/ttt.k   learning file

NAME

wump - the game of hunt-the-wumpus

SYNOPSIS

/usr/games/wump

DESCRIPTION

The *wump* program plays the game of "Hunt the Wumpus." A Wumpus is a creature that lives in a cave with several rooms connected by tunnels. You wander among the rooms, trying to shoot the Wumpus with an arrow, meanwhile avoiding being eaten by the Wumpus and falling into Bottomless Pits. There are also Super Bats which are likely to pick you up and drop you in some random room.

The program asks various questions which you answer one per line; it gives a more detailed description if you want.

This program is based on one described in *People's Computer Company*, 2, 2 (November 1973).

BUGS

It will never replace Adventure.

NAME

intro - introduction to special files

SYNOPSIS

#include <sys/socket.h>
#include <netinet/ip_str.h>

DESCRIPTION

This section describes various special files that refer to specific hardware peripherals and CTIX System device drivers, including networking protocol drivers. Features common to a set of protocols are documented as a protocol family.

HARDWARE ENTRIES

The names of these 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 CTIX system device driver are discussed where applicable.

PROTOCOL FAMILY ENTRIES

A protocol family provides basic services to the protocol implementation to allow it to function within a specific network environment. These services may include packet fragmentation and reassembly, routing, addressing, and basic transport. A protocol family can support multiple methods of addressing, though the current protocol implementations do not. A protocol family is normally comprised of a number of protocols, one per *socket*(2) type. It is not required that a protocol family support all socket types. A protocol family can contain multiple protocols supporting the same socket abstraction.

A protocol supports one of the socket abstractions detailed in *socket*(2). A specific protocol can be accessed by creating a socket of the appropriate type and protocol family, by requesting the protocol explicitly when creating a socket, by executing the appropriate TLI primitives, or by opening the associated STREAMS device.

PROTOCOL ENTRIES

The system currently supports the DARPA Internet protocols. Raw socket interfaces are provided to the IP protocol layer of the DARPA Internet and to ICMP protocol. Consult the appropriate manual pages in this section for more information.

ROUTING IOCTLS

The network facilities provide limited packet routing. A simple set of data structures comprise 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. A user process, the routing daemon, maintains this data base with the aid of two socket-specific *ioctl*(2) commands, SIOCADDRT and SIOCDELRT. The commands allow the addition and deletion of a single routing table entry, respectively. Only the super-user can carry out routing table manipulations.

A routing table entry has the following form, as defined in <*net/route.h*>:

```
struct rtentry {
        u_long   rt_hash;
        struct   sockaddr rt_dst;
        struct   sockaddr rt_gateway;
        short    rt_flags;
        short    rt_refcnt;
        u_long   rt_use;
        struct   ifnet *rt_ifp;
};
```

where *rt_flags* is defined as follows:

```
#define  RTF_UP       0x1   /* route usable */
#define  RTF_GATEWAY  0x2   /* destination is a gateway */
#define  RTF_HOST     0x4   /* host entry (net otherwise) */
#define  RTF_DYNAMIC  0x10  /* created dynamically
                               (by redirect) */
```

Routing table entries are of three general types: those for a specific host, those for all hosts on a specific network, and those for any destination not matched by entries of the first two types (a wildcard route). When the system is booted and addresses are assigned to the network interfaces, each protocol family installs a routing table entry for each interface when it is ready for traffic. Normally the protocol specifies the route through each interface as 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 is requested to address the packet to the gateway listed in the routing entry (that is, the packet is forwarded). Some routing entries specify a connection requiring some form of dialing; see *slipd*(1M).

Routing table entries installed by a user process cannot 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 routing entry is marked down and removed from the routing table, but the resources associated with it are be reclaimed until all references to it are released. The routing code returns EEXIST if requested to duplicate an existing entry, ESRCH if requested to delete a non-existent entry, or ENOSR if insufficient resources were available to install a new route. User processes read the routing tables through the */dev/kmem* device. The *rt_use* field contains the number of packets sent along the route.

When routing a packet, the kernel first attempts to find a route to the destination host. Failing that, a search is made for a route to the network of the destination. Finally, any route to a default (''wildcard'') gateway is chosen. If multiple routes are present in the table, the first route found is used. If no entry is found, the destination is declared to be unreachable.

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.

INTERFACE IOCTLS

Each network interface in a system corresponds to a path through which messages can be sent and received. A network interface usually has a hardware device associated with it, although certain interfaces such as the loopback interface, *lo*(7), do not.

The following *ioctl* calls can be used to manipulate network interfaces. The *ioctl* is made on a socket (typically of type SOCK_DGRAM) in the desired "communications domain" [see *protocols*(4)]. Unless specified otherwise, the request takes an *ifrequest* structure as its parameter. This structure has the following form:

```
struct  ifreq {
        char    ifr_name[16]; /* name of interface (e.g. ec0) */
        union {
                struct  sockaddr ifru_addr;
                struct  sockaddr ifru_dstaddr;
                struct  sockaddr ifru_broadaddr;
                short   ifru_flags;
                int     ifru_metric;
        } 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_broadaddr ifr_ifru.ifru_broadaddr
                /* broadcast address */
#define ifr_flags ifr_ifru.ifru_flags   /* flags */
```

```
#define ifr_metric ifr_ifru.ifru_metric /* routing metric */
};
```

SIOCSIFADDR
Set interface address for protocol family. Following the address assignment, the "initialization" routine for the interface is called.

SIOCGIFADDR
Get interface address for protocol family.

SIOCSIFDSTADDR
Set point-to-point address for protocol family and interface.

SIOCGIFDSTADDR
Get point-to-point address for protocol family and interface.

SIOCSIFBRDADDR
Set broadcast address for protocol family and interface.

SIOCGIFBRDADDR
Get broadcast address for protocol family and interface.

SIOCSIFFLAGS
Set interface flags field. If the interface is marked down, any processes currently routing packets through the interface are notified; some interfaces can be reset so that incoming packets are no longer received. When marked up again, the interface is reinitialized.

SIOCGIFFLAGS
Get interface flags.

SIOCSIFMETRIC
Set interface routing metric. The metric is used only by user-level routers.

SIOCGIFMETRIC
Get interface metric.

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 contains the length, in bytes, of the configuration list.

```
/* Structure used in SIOCGIFCONF request.
 * Used to retrieve interface configuration
 * for machine (useful for programs which
 * must know all networks accessible). */
```

```
struct    ifconf {
          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 */
};
```

STREAMS IOCTL INTERFACE

Socket *ioctl* calls can also be issued using STREAMS file descriptors. The standard *strioctl* structure is used, with the *ic_cmd* field containing the socket *ioctl* code (from <sys/socket.h>) and the *ic_db* field pointing to the data structure appropriate for that *ioctl*.

Options management is performed by using the TLI primitives and the following structure, which contains the arguments to the "*sockopts*" calls:

```
struct optdesc {
        int level;      /* Protocol Level Affected */
        int optname;    /* option name to modify */
        int value;      /* value set or retrieved */
};
```

SEE ALSO

routed(1M), ioctl(2), socket(2).
CTIX Network Administrator's Guide.
CTIX Network Programmer's Primer.
UNIX System V Release 3.2 Network Programmer's Guide.

NOTE

CTIX Internetworking manual pages frequently cite appropriate RFCs (Requests for Comments). RFCs can be obtained from the DDN Network Information Center, SRI International, Menlo Park, CA 94025.

NAME

arp - Address Resolution Protocol

DESCRIPTION

ARP is a protocol used to dynamically map between DARPA Internet and 10Mb/s Ethernet addresses. It is used by all the 10Mb/s Ethernet interface drivers running the Internet protocols.

ARP caches Internet-Ethernet address mappings. When an interface requests a mapping for an address not in the cache, ARP queues the message which 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 queues at most one packet while waiting for a mapping request to be responded to; only the most recently ''transmitted'' packet is kept. The ARP protocol is implemented by a STREAMS driver to do the protocol negotiation, and a separate STREAMS module to do the address translation.

To facilitate communications with systems which do not use ARP, *ioctl* s are provided to enter and delete entries in the Internet-to-Ethernet tables. Usage:

```
#include <sys/ioctl.h>
#include <sys/socket.h>
#include <net/if.h>
struct arpreq arpreq;

ioctl(s, SIOCSARP, (caddr_t)&arpreq);
ioctl(s, SIOCGARP, (caddr_t)&arpreq);
ioctl(s, SIOCDARP, (caddr_t)&arpreq);
```

Each *ioctl* takes the same structure as an argument. SIOCSARP sets an ARP entry, SIOCGARP gets an ARP entry, and SIOCDARP deletes an ARP entry. These *ioctls* can be applied to any socket descriptor *s*, but only by the super-user. The *arpreq* structure is as follows:

```
    /* 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_COM           0x02         /* completed entry */
                                        /* (arp_ha valid) */
#define  ATF_PERM          0x04         /* permanent entry */
```

```
#define  ATF_PUBL           0x08    /* 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 which may be written are ATF_PERM, ATF_PUBL and ATF_USETRAILERS. ATF_PERM causes the entry to be permanent if the *ioctl* call succeeds. The peculiar nature of the ARP tables may cause the *ioctl* to fail if more than 8 (permanent) Internet host 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 which 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 that responds to an ARP mapping request for the local host's address).

SEE ALSO

arp(1M), ifconfig(1M), en(7), inet(7).
RFC 826, RFC 893.

DIAGNOSTICS

duplicate IP address!! sent from ethernet address: %x:%x:%x:%x:%x:%x. ARP has discovered another host on the local network which responds to mapping requests for its own Internet address.

NAME

clone - open any minor device on a STREAMS driver

DESCRIPTION

The *clone* driver is a STREAMS software driver that finds and opens an unused minor device on another STREAMS driver. The minor device passed to *clone* during the open is interpreted as the major device number of another STREAMS driver for which an unused minor device is to be obtained. Each such open results in a separate *stream* to a previously unused 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*.

The *clone* generates an ENXIO error, without opening the device, if the minor device number provided does not correspond to a valid major device, or if the driver indicated is not a STREAMS driver.

SEE ALSO

log(7).
UNIX System V Release 3.2 Streams Programmer's Guide.

CAVEATS

The *clone* interface cannot perform multiple opens of one minor device. Executing *stat*(2) on the file system node for a cloned device yields a different result from executing *fstat*(2) using a file descriptor obtained from opening the node.

NAME

console - console terminal

DESCRIPTION

The special file **/dev/console** designates a standard destination for system diagnostics. The kernel writes its diagnostics to this file, as does any user process with messages of systemwide importance. If **console** is associated with a physical terminal (configured with the kernel debugger), console messages appear on that terminal.

Note that *inittab*(4) does not normally post a *getty* on **console**; **console** might become associated with a terminal that already is a login terminal.

Console messages are saved in a circular buffer. Reading **console** retrieves the messages and removes them from the buffer. Unless CTIX is configured with the kernel debugger, **console** is not associated with a terminal; console messages are written to **/etc/log/confile.**

If CTIX is configured with the kernel debugger [see *dbconsole*(1M), *uconf*(1M), *system*(4), and the **/etc/drvload** file], the terminal associated with the console (by default, **tty000**) receives console messages, and a Control-B on that terminal starts the kernel debugger.

The size of the console circular buffer is configured by using the *config*(1M) parameter **cbufsz.** The default is 4096 bytes.

The following *ioctl*(2) commands are accepted:

ioctl(fd, CONERR);

fd must be open to **console.** All console output is to be duplicated in the error message queue. See *err*(7).

ioctl(fd, CONBUF);

fd must be open to **console.** No console output is to be duplicated in the error message queue. This is the initial condition.

ioctl(fd, CON_SET, port)

fd must be open to *console. port* is the minor device number of the RS-232 line that will be the new debugger console; *port* must be a valid RS-232 channel. The function returns the number of the new debugger console port.

ioctl(fd, CON_LOC)

fd must be open to *console.* The function returns the number of the current debugger console port.

FILES

/dev/console
/etc/log/confile

SEE ALSO

conlocate(1M), syslocal(2).

WARNING

Normal system processing is suspended while the kernel debugger is active.

NAME

disk - general disk driver

SYNOPSIS

#include <sys/types.h>
#include <sys/gdisk.h>
#include <sys/gdioctl.h>

DESCRIPTION

The CTIX special files **/dev/rdsk/c0d0s0** through **/dev/rdsk/c*x*d*x*s*x*** and **/dev/dsk/c0d0s0** through **/dev/dsk/c*x*d*x*s*x*** refer to CTIX device names and slices, where **c***x* is the controller number, **d***x* is the drive number, s*x* is the slice number, and *x* is a hexadecimal digit. An **r** in the name indicates the character (raw) interface.

A disk is formatted with 512-byte physical sectors. Logical block zero contains the *Volume Home Block* (VHB), which describes the disk. The VHB is structured to use two physical sectors as one logical block (1024 bytes).

The following structure defines the Volume Home Block:

```
struct vhbd {
    uint  magic;            /* S/MT disk format code */
    int   chksum;           /* adjustment so 32-bit sum starting
                               from magic for 1K bytes sums to -1 */
    struct gdswprt dsk;     /* specific description of this disk */
    struct partit partab[MAXSLICE]; /* partition table */
    struct resdes {         /* reserved area special files */
          daddr_t blkstart; /* start logical block # */
          ushort nblocks;  /* length in logical blocks
                               (zero implies not present) */
    } resmap[8];
    /*    resmap consists of the following entries:
     *                      loader area
     *                      bad block table
     *                      dump area
     *                      down load image file
     *                      Bootable program,
     *                      size determined by a.out format. nblocks=1.
    */
    char  fpulled;          /* dismounted last time? */
    long  time;             /* time last came on line */
    struct gdswprt2 dsk2; /* Drive specific parameters */
    char  minires[38];      /* for future mini/mitl frame enhancements */
```

```
    char   sysres[292];      /* custom system area */
    struct mntnam mntname[MAXSLICE];
                             /* names for auto mounting; null
                              * string means no auto mount
                              * not used in mitiframe */
    char   userres[256];     /* user area */
};
struct gdswprt {
    char     name[6];   /* printf name */
    ushort   cyls;      /* the number of cylinders for this disk */
    ushort   heads;     /* number of heads per cylinder */
    ushort   psectrk;   /* number of physical sectors per track */
    ushort   pseccyl;   /* number of physical sectors */
                        /* per cylinder */
    char     flags;     /* floppy density and high tech drive flags */
    char     step;      /* stepper motor rate to controller - ST506 only */
    ushort   sectorsz;  /* size of physical sectors (in bytes) */
};
struct gdswprt2 {
    short    wpccyl;    /* value to program for RWC/WPC - ST506 only */
    ushort   enetaddr[3];/* Ethernet station address -
                        * MiniFrame only */
    unchar   gap1;      /* Gap size on SMD drives */
    unchar   gap2;
    char     filler[28];
};
#define  sparesec    gap1    /* spare sectors per track  */
#define  sparecyl    gap2    /* spare tracks per cylinder */
#define  scinterleave wpccyl /* interleave factor  */

struct partit{
    union {
        uint strk;/* start track number (new style) */
        struct {
            ushort strk;/* start track # */
            ushort nsecs;/* # logical blocks available to user */
        } old;
    } sz;
};
```

If a VHB is valid, *magic* is equal to VHBMAGIC and the 32-bit sum of the VHB's bytes is 0xFFFFFFFF (-1); *chksum* is the adjustment that makes the sum come out right.

The *dsk* structure describes the peculiarities of the disk, including deliberate deviations from the system standard. The *dsk.flags* field is the bitwise OR of zero or more of the following constants:

HITECH (ST506 only) If on, head select bit 3 is valid; if off, reduced write current is valid.

NEWPARTTAB If off, the old style slice (partition) table is in use; if on, the new style slice table is in use.

RWCPWC (ST506 only) If on, set reduced write current/write precompensation.

HITECH selects write precompensation.

FORMATEXTRA If on, the SMD drive is formatted with an extra sector on each track. (This sector is ignored by CTIX but is required for some disk drives, notably the Eagle-XP.)

The *dsk.step* field specifies a stepper motor rate for the ST506; use 14 in this field.

The *partab* structure divides the disk into slices (partitions).

The *fpulled* field indicates whether an exchangeable disk was properly removed from the drive. The system sets this field to 1 when the disk is inserted in the drive. To clear *fpulled*, run *dismount*(1M).

The *mntname*, *minires*, and *userres* arrays are reserved for future use.

The *resmap* array describes the files that share Slice 0 with the Volume Home Block. Provision is made for eight such files, but only five have been assigned slots in *resmap*. Each *resmap* entry gives the starting location (logical block number) and length (logical blocks). A length of zero indicates that the file is not provided. The first five entries in *resmap* describe the following:

1. The loader. When the system is reset or turned on, the boot prom loads the loader into the loader address and jumps execution to it. The function of the loader is to search for and load a program that will boot the system.

On the S/640 and S/480, the loader searches the onboard tape, onboard (ST506) disks 0, 1, and 2, the VME, and the SCSI disks, in that order.

On each disk, the loader checks for a CTIX kernel, which must be a CTIX executable object file called **/unix** in the file system in slice 1. When the loader locates an appropriate program, it preserves the crash dump table, loads the program it found at the address it was linked at (0x0 if unknown) and executes it. If no disk contains an appropriate file, the loader continues searching until an appropriate disk is inserted.

2. The bad block table, which always begins at logical block 1 of the disk. Each logical block in the bad block table consists of a four-byte checksum followed by 127 bad block cells. The checksum is a value that makes the 32-bit sum of the logical block be 0xFFFFFFFF (-1). A bad block cell is defined by the following structure.

```
struct bbcell {
    ushort cyl;      /* the cylinder of the bad block */
    ushort badblk;   /* the physical sector address of the
                        bad block within the cylinder cyl */
    ushort altblk;   /* track number of alternate */
    ushort nxtind;   /* index into the cell array for next
                        bad block cell for this cylinder */
};
```

A single sequence of numbers, starting from zero, identifies the checksums and cells. For non-SCSI disks, in each cell in use, *cyl* identifies a cylinder that contains the bad block; *badblk* is the physical block offset within the cylinder of the bad block; *altblk* identifies the track that contains the alternate block; *nextind* (not used in S/MT) identifies the next cell for a bad block on the same cylinder or, if this is the last bad block, is zero.

SCSI disks perform their own bad block housekeeping. The bad block table contains only blocks that CTIX cannot read. At the next attempt to write to the bad block, CTIX issues a reassign block command to the SCSI drive. The drive then performs the bad block mapping for that sector, and the sector number is removed from the bad block table.

3. The dump area. After a reset or system crash, the boot prom dumps processor registers, the memory map, a crash dump block, and the contents of physical memory, until it runs out of room in the dump area.

4. The download image area. The download images are described by a table at the beginning of the area. The area is described by the following array:

```
struct dldent {
        short d_strt; /* block displacement from download index */
        short d_sz; /* # of blocks for this entry */
};
```

The image number is the index for *dldent*. The *d_strt* field is the offset in bytes of the image from the beginning of the download image area; *d_sz* is the size in bytes of the image.

Slice 0 is called the Reserved Area. Only the Volume Home Block and the files described by *resmap* can be in the Reserved Area. A formatted disk used by a working system certainly has at least one more slice.

The *ioctl* system calls use the following structure:

```
struct gdioctl {
   ushort  status;                /* status */
   struct  gdswprt params;        /* description of the disk */
   struct  gdswprt2 params2;      /* more description of the disk */
   short   ctrltyp;               /* the type of disk controller */
   short   driveno;
};
```

where *status* Is the bitwise OR of the following constants:

VALID_VHB A valid Volume Header Block has been read.

DRV_READY The disk is on line.

PULLED Last removal of disk from drive was not preceded by proper dismount.

params is a *gdswprt* structure, the same type used in the volume header block.

dsktype is equal to one of the following:

GD_WD1010	for Western Digital 1010 ST506 Controller
GD_WD2010	for Western Digital 2010 ST506 Controller
GD_RAMDISK	for RAM Disk Emulator
GD_SMD3200	for Interphase SMD3200 disk controller
GD_SCSI	for SCSI disk controller

CTIX understands the following disk *ioctl* calls:

ioctl(fd, GDIOCTYPE, 0)

Returns GDIOC if *fd* is a file descriptor for a disk special file.

ioctl(fd, GDGETA, gdctl_ptr)

gdctl_ptr is a pointer to a *gdioctl* structure. *ioctl* fills the structure with information about the disk.

ioctl(fd, GDSETA, gdctl_ptr)

gdctl_ptr is a pointer to a gdioctl structure. *ioctl* passes the description of the disk to the disk driver. This is primarily meant for reading disks created by other kinds of computers.

ioctl(fd, GDFORMAT, ptr)

ptr points to formatting information. The disk driver formats a track.

ioctl(fd, GDDISMNT)

ioctl informs the driver that the user intends to remove the disk from the drive. When this system call successfully returns, the driver has flushed all data in the buffer cache and waited for all queued transfers to complete. The last transfer is to write out the Volume Home Block with the *fpulled* flag cleared. Once this call returns, the drive is inaccessible until a new disk is inserted.

ioctl(fd, GDPASSTHRU, arg)

arg points to a disk driver-specific command block; *gd* passes the command to the specific disk driver untouched, and the disk driver performs the specific command.

SEE ALSO

iv(1), mknod(1M), ioctl(2).
S/Series CTIX Administrator's Guide.

NAME

drivers - loadable device drivers

DESCRIPTION

A loadable driver is equivalent to a fixed, linked-in device driver. It has access to all kernel subroutines and global data. After it is loaded, it is effectively part of the running kernel.

Differences between loadable and ordinary drivers involve their driver ID, init routine, release routine, and interrupt processing.

Driver ID

All drivers have a driver ID. Preloaded drivers have a driver ID of 0. Loaded drivers are given an ID when they allocate virtual space. The driver ID is automatically set when the driver is linked. The ID should never be modified by the driver itself; the ID is used to identify the driver to the system when making certain requests.

Init Routine

Loadable drivers may have an init routine that is executed when the driver is bound, and a release routine that is executed when the driver is unbound [see *lddrv*(1M) for a description of driver allocation and bind operations]. Init routines check for the existence of hardware, initialize the hardware, put the interrupt service routine for the hardware into the interrupt chain, and do other similar tasks.

Release Routine

Release routines make sure the device or driver is idle, turn off the device, take the interrupt service routine out of the interrupt chain, and similar tasks. A typical action for a release routine to take when the device *is not* idle is to set an error code in **u.u_error** and return.

Interrupt Processing

For details about CTIX interrupt processing, refer to the *Writing MightyFrame Device Drivers* manual.

EXAMPLE

```
/* init, release, interrupt service routines */
/* for loadable device xyzzy */

#include <sys/drv.h>

#define XYZ_VECNO     0x60     /* interrupt vector number */
#define XYZ_BUSY      1        /* flags */
#define XYZ_OPEN      2
int xyzzint();                 /* interrupt service routine */

extern int DFLT_ID;
```

```
static int Drv_id = &DFLT_ID;          /* set drive ID */
int xy_base;
int xy_flags;

xy_init()
{
        if (set_vec(Drv_id, XYZ_VECNO, xyzzyint) < 0)
        {
                u.u_error = EBUSY;
                return;
        }
        .
        .
        <do hardware initialization>
        .
}
xy_release()
{
        if (xy_flags & (XY_BUSY | XY_OPEN))
        {
                u.u_error = EBUSY;
                return;
        }
        .
        .
        <turn off device>
        .
        reset_vec (Drv_id, XYZ_VECNO);
}
xyzzyint()
{
        .
        <clear interrupt>
        .
        <process interrupt>
        .
}
```

FILES

/etc/master

SEE ALSO

lddrv(1M), master(4).
Writing MightyFrame Device Drivers.

NAME

en - Ethernet Processor

DESCRIPTION

The *en* interface provides access to a 10 Mb/s Ethernet network through a CMC ENP-10 Ethernet Processor or a Convergent Technologies Ethernet/RS-232 Combo or SCSI/LAN Board.

Each of the host's network addresses is specified at boot time with an SIOCSIFADDR *ioctl*. An *en* interface usually uses the address resolution protocol described in *arp*(7) to dynamically map between Internet and Ethernet addresses on the local network.

The interface normally tries to use a "trailer" encapsulation to minimize copying data on input and output. The use of trailers is negotiated with ARP. This negotiation may be disabled, on a per-interface basis, by setting the IFF_NOTRAILERS flag with an SIOCSIFFLAGS *ioctl*.

Two special *ioctls* have been defined for retrieving information about Ethernet interfaces. These *ioctls* can be executed by using either a socket file descriptor [returned by *socket*(2)], or a file descriptor [returned by *open*(2) or *t_open*(3)]. When using a socket file descriptor, the Ethernet device unit number is determined by the interface name passed in the *ifreq* structure; for example, unit zero is **en0**. For TLI calls, the clonable device **/dev/enet** is opened and the **IF_UNITSEL** *ioctl* is used to specify the unit number. The data argument to the *ioctl* call is an *int* containing the desired unit number. The **I_STR** *ioctl* is then used for the main call.

The **SIOCGENADDR** *ioctl* returns the six byte hardware Ethernet address being used by an interface. Using a socket file descriptor, the address is returned in the *ifr_enaddr* element of *ifreq* structure. Using a TLI file descriptor, the *ic_db* field of the *strioctl* structure should point directly to six bytes of storage, which contains the address on return.

The **SIOCGENPSTATS** *ioctl* returns various statistics about the interface. The data returned is 22 longs (88 bytes) long, consisting of an *enp_stats* structure followed by three longs as described below. Using a socket file descriptor, the data pointer argument to the *ioctl* call should point to an *ifreq* structure, whose *ifr_data* element points to the storage. Using a TLI file descriptor, the *ic_dp* pointer in the *strioctl* structure should point directly to the storage.

```
struct enp_stats {
        uint    es_transmit;    /* number of good transmits */
        uint    es_retry_many;  /* # multiple retries reported */
        uint    es_retry_one;   /* # single retries reported */
```

```
        uint    es_retry_fail;   /* # failed retries */
        uint    es_defer;        /* # deferrals reported */
        uint    es_tbuf_err;     /* # of transmit BUF errors */
        uint    es_urun;         /* # SILO underruns */
        uint    es_late_coll;    /* # late collisions */
        uint    es_carr_loss;    /* # carrier losses */
        uint    es_babl;         /* # babbling tranmitter errors */
        uint    es_coll;         /* # collision errors */
        uint    es_mem;          /* # memory errors on transmit */
        uint    es_receive;      /* # good packets received */
        uint    es_miss;         /* # missed packets reported */
        uint    es_crc;          /* # CRC errors reported */
        uint    es_fram;         /* # framing errors reported */
        uint    es_rbuf_err;     /* # receive BUF errors */
        uint    es_orun;         /* # SILO overruns */
        uint    es_rmem;         /* # memory errors on receive */
};

        extra[0] = MemoryErrors;   /* receive mblock not available */
        extra[1] = TXAvailErrors;  /* LANCE transmit buffer/*
                                   /* not available */
        extra[2] = RingPutErrors;  /* receive queue full */
```

SEE ALSO

intro(7), inet(7), arp(7).

DIAGNOSTICS

Couldn't get interrupt vector for en%d

The system interrupt vector table was full.

en%d doesn't respond to initialization

The onboard software does not respond to its initialization interrupt.

NAME

err - error-logging interface

DESCRIPTION

Minor device 0 of the *err* driver is the interface between a process and the system's error-record collection routines. The driver can be opened only for reading by a single process with super-user permissions. Each read causes an entire error record to be retrieved and removed; the record is truncated if the read request is for less than the record's length.

An appropriate command to the console sends console information to the error record queue. See *console*(7).

FILES

/dev/error special file

SEE ALSO

errdemon(1M), console(7).

NAME

icmp - Internet Control Message Protocol

SYNOPSIS

#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_RAW, proto);

DESCRIPTION

ICMP is the error and control message (or device) protocol used by IP and the Internet protocol family. It may be accessed through a "raw socket" for network monitoring and diagnostic functions. The *proto* parameter to the socket call to create an ICMP socket is obtained from *getprotobyname* [See *getprotoent*(3).] ICMP sockets are connectionless, and are normally used with the *sendto* and *recvfrom* calls; the *connect*(2) call may also be used to fix the destination for future packets [in which case the *read*(2) or *recv*(2) and *write*(2) or *send*(2) system calls may be used].

Outgoing packets automatically have an IP header prepended to them (based on the destination address). Incoming packets are received with the IP header and options intact.

FILES

/dev/inet/icmp

SEE ALSO

send(2), recv(2), intro(7), inet(7), ip(7).

DIAGNOSTICS

A socket operation may fail with one of the following errors returned:

[EISCONN] when trying 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] when trying to send a datagram, but no destination address is specified, and the socket hasn't been connected;

[ENOSR] when the system runs out of memory for an internal data structure;

[EADDRNOTAVAIL] when an attempt is made to create a socket with a network address for which no network interface exists.

NAME

inet - Internet protocol family

SYNOPSIS

#include <sys/types.h>
#include <sys/in.h>

DESCRIPTION

The Internet protocol family is a set of protocols using the *Internet Protocol* (IP) network layer and the Internet address format. The Internet family provides protocol support for the SOCK_STREAM, SOCK_DGRAM, and SOCK_RAW socket types; the SOCK_RAW interface provides access to the IP protocol.

ADDRESSING

Internet addresses are four-byte quantities, stored in network standard format. The include file *<sys/in.h>* defines this address as a discriminated union.

Sockets bound to the Internet protocol family use the following addressing structure:

```
struct sockaddr_in {
        short     sin_family;
        u_short   sin_port;
        struct    in_addr sin_addr;
        char      sin_zero[8];
};
```

Sockets may be created with the local address INADDR_ANY to affect wildcard matching on incoming messages. The address in a *connect*(2) or *sendto* [see *send*(2)] call may be given as INADDR_ANY to mean "this host." The distinguished address INADDR_BROADCAST is allowed as a shorthand for the broadcast address on the primary network if the first network configured supports broadcast.

PROTOCOLS

The Internet protocol family is comprised of the IP transport protocol, Internet Control Message Protocol (ICMP), Transmission Control Protocol (TCP), and User Datagram Protocol (UDP). TCP is used to support the SOCK_STREAM abstraction; UDP is used to support the SOCK_DGRAM abstraction. A raw interface to IP is available by creating an Internet socket of type SOCK_RAW. The ICMP message protocol is accessible from a raw socket.

The 32-bit Internet address contains both network and host parts. It is frequency-encoded; the most-significant bit is clear in Class A addresses, in which the high-order 8 bits are the network number. Class B addresses use the high-order 16 bits as the network field, and Class C addresses have a 24-bit network part. Sites with a cluster of local networks and a connection to the DARPA Internet may chose to use a single network number for the cluster; this is done by using subnet addressing. The local (host) portion of the address is further subdivided into subnet and host parts. Within a subnet, each subnet appears to be an individual network; externally, the entire cluster appears to be a single, uniform network requiring only a single routing entry. Subnet addressing is enabled and examined by the following *ioctl*(2) commands on a datagram socket in the Internet "communications domain"; they have the same form as the SIOCIFADDR command [see *intro*(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.

SEE ALSO

ioctl(2), socket(2), intro(7), icmp(7), ip(7), tcp(7), udp(7).
CTIX Network Administrator's Guide.

CAVEAT

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);

DESCRIPTION

IP is the network layer protocol used by the Internet protocol family. Options may be set at the IP level when using higher-level protocols that are based on IP (such as TCP and UDP). It may also be accessed through a "raw socket" or device when developing new protocols or special purpose applications.

A single generic option is supported at the IP level, IP_OPTIONS, that may be used to provide IP options to be transmitted in the IP header of each outgoing packet. Options are set with *setsockopt* and examined with *getsockopt* [see *getsockopt*(2)]. The format of IP options to be sent is that specified by the IP protocol specification, with one exception: the list of addresses for Source Route 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.

Raw IP sockets are connectionless, and are normally used with the *sendto* and *recvfrom* calls; the *connect*(2) call may also be used to fix the destination for future packets (in which case the *read*(2) or *recv*(2) and *write*(2) or *send*(2) system calls may be used).

If *proto* is 0, the default protocol IPPROTO_RAW is used for outgoing packets, and only incoming packets destined for that protocol are received. If *proto* is non-zero, that protocol number will be used on outgoing packets and to filter incoming packets.

Outgoing packets automatically have an IP header prepended to them (based on the destination address given and the protocol number the socket is created with). Incoming packets are received with IP header and options intact.

SEE ALSO

getsockopt(2), send(2), recv(2), intro(7), icmp(7), inet(7).
CTIX Network Administrator's Guide.

DIAGNOSTICS

A socket operation may fail with one of the following errors returned:

[EISCONN] when trying 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] when trying to send a datagram, but no destination address is specified, and the socket has not been connected

[ENOSR] when the system runs out of memory for an internal data structure

[EADDRNOTAVAIL]
when an attempt is made to create a socket with a network address for which no network interface exists

The following errors specific to IP 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.

NAME

ipt - interface for Interphase V/TAPE 3200 half-inch tape controller

DESCRIPTION

The *ipt* interface provides access to up to eight Interphase half-inch tape drives per tape controller; Note that there can be only one Interphase tape controller.

By default, the major device number is 19. Bits 4 through 7 of the minor device number specifies the tape density, rewind option, and whether it is ioctl only, as follows (where "not 4" indicates that bit 4 is *not* set, and "4" indicates that bit 4 *is* set, and so on for each bit):

not 4 and not 5	medium density
4 and not 5	low density
not 4 and 5	high density
4 and 5	use last density
6	rewind on close
7	ioctl only

A standard naming convention for tape devices has been adopted: **/dev/rmt/c1d#**[*dens*][**n**], where

d# Indicates the drive number on that controller

dens Indicates the density (**h**, **m**, or **l** **h** (high) specifies 6250 bpi; **m** (medium) specifies 1600 bpi; and **l** (low) is normally 800 bpi.

Using a tape device name that does not include a density specification (**h**, **m**, or **l**) implies that no special density, gap, or speed command should be issued by the driver. [See *tapeset*(1M) and *tapedrives*(4).] If a drive uses formatter commands for changing density, gap or speed, the device without density specification does not generally change the current setting. If a drive uses the J connector lines to select density, gap, or speed, the drive is usually switched to low density, default gap, and slow speed. The *ipt* device is generally most useful with the *dumb* tapedrive type (in **/etc/tapedrives**), when the characteristics of the drive are not known.

n Indicates no rewind on close.

The special file name **/dev/rmt/c1d#c** can be used to set drive parameters with the *tapeset*(1M) command.

Tape files are separated by file marks. Closing a file open for writing writes two tape marks; if the device is no-rewind, the tape is left positioned between the two tape marks.

Each *read* or *write* reads or writes the next physical record. The size of a *write* determines the size of the next record. A *read* need not match the size of the record. If a *read* requests more bytes than available, the *read* returns the number of bytes in the record. If a *read* requests fewer bytes than available, the *read* returns the requested number of bytes, and the remainder of the record is skipped. Seeks are ignored. Reading a file mark produces a zero-length read and leaves the tape positioned before the mark. The program must, therefore, issue an *ioctl* call to skip the file mark (alternatively, the program can close and re-open the no-rewind device). Attempting to read a bad record leaves the tape positioned after the faulty record.

As shown below, *ioctl*(2) supports the following commands for half-inch tape:

```
#include <sys/iptioctl.h>
ioctl(fildes, cmd, arg)
```

where *cmd* is one of the following:

IPTIOCTYPE	Always return IPTIOC
IPTGETA	Read the controller's current operational parameters into an *iptinfo* structure pointed to by *arg*.
IPTSETA	Initialize the controller with the operational parameters specified in the *iptinfo* structure pointed to by *arg*.
IPTCMD	Specify a command to the tape controller as specified in *arg*. Legal values of *arg* follow:

REWIND	Issue a rewind command.
WFM	Issue a write file mark command.
RFM	Issue a read file mark command.
ERASE	Issue an erase command.
CLERR	Clear software error state.

IPTFMTCMD	Issue a command directly to a tape formatter. The command byte pointed to by *arg* is sent to the specified drive.

In the following commands, *arg* must be a pointer to an *iptinfo* structure defined as follows:

```
struct iptinfo {
    uint maxblksize;        /* max block size */
    uint parms0;            /* not used */
    uint parms1;            /* not used */
    uint parms2;            /* used, bits defined as: */

#define DSBOK  0x01    /* Can set density */
#define DSB    0x02    /* 0: lines, 1: fmt cmd */
#define DSBL   0x04    /* 0: J1_36, 1: J2_50 */
#define DSBFLGS (DSBOK | DSB | DSBL)

#define SSBOK  0x08    /* Can set speed */
#define SSB    0x10    /* 0: lines, 1: fmt cmd */
#define SPD    0x20    /* 0: low,   1: high */
#define SPDL   0x40    /* 0: J1_36, 1: J2_50 */
#define SPDFLGS (SSBOK | SSB | SPD | SPDL)

#define GSBOK  0x80    /* Can set gap */
#define GSB    0x100   /* 0: lines, 1: fmt cmd */
#define LGAP   0x200   /* 0: normal,1: extended */
#define LGAPL  0x400   /* 0: J1_36, 1: J1_44 */
#define GAPFLGS (GSBOK | GSB | LGAP | LGAPL)

    unchar   hspeed;    /* high speed code */
    unchar   lspeed;    /* low speed code */

    unchar   dgap;      /* default gap code */
    unchar   xgap;      /* extended gap code */

    unchar   denslow;   /* low density code */
    unchar   densmed;   /* medium density code */
    unchar   denshigh;  /* high density code */

    ushort   status;    /* drive status */
    unchar   ctlr;      /* controller number */
    unchar   unit;      /* unit number  */
};
```

FILES

/dev/rmt/c1d[0-7]	don't issue density, gap, or speed command device
/dev/rmt/c1d[0-7]c	ioctl-only device
/dev/rmt/c1d[0-7]l	low-density device
/dev/rmt/c1d[0-7]m	medium-density device
/dev/rmt/c1d[0-7]h	high-density device
/dev/rmt/c1d[0-7]ln	low-density device (no rewind)
/dev/rmt/c1d[0-7]mn	medium-density device (no rewind)
/dev/rmt/c1d[0-7]hn	high-density device (no rewind)

SEE ALSO

brc(1M), config(1M), tapeset(1M), ioctl(2), lddrv(1M), system(4), qic(7), stape(7), vme(7).
MightyFrame VME Half-Inch Tape Controller Card Manual.

NAME

lo - software loopback network interface

SYNOPSIS

pseudo-device loop

DESCRIPTION

The *loop* interface is a software loopback mechanism that can be used for performance analysis, software testing, and/or local communication. As with other network interfaces, the loopback interface must have network addresses assigned for each address family with which it is to be used. These addresses can be set or changed by using the SIOCSIFADDR ioctl. The loopback interface should be the last interface configured, as some protocols use the order of configuration as an indication of priority. The loopback should *never* be configured first unless no hardware interfaces exist.

SEE ALSO

inet(7).

NAME

log - interface to STREAMS error logging and event tracing

DESCRIPTION

The *log* driver is a STREAMS software device driver that provides an interface for the STREAMS error logging and event tracing processes [*strerr*(1M), *strace*(1M)]. The *log* driver 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 error logger, a trace logger, or processes that need to submit their own *log* messages.

Kernel Interface

The *log* driver's messages are generated within the kernel by calls to the function *strlog*:

```
strlog(mid, sid, level, flags, fmt, arg1, ...)
short mid, sid;
char level;
ushort flags;
char *fmt;
unsigned arg1;
```

Required definitions are contained in **<sys/strlog.h>** and **<sys/log.h>**. *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_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

The *log* driver is opened through 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 or trace logger through a STREAMS I_STR *ioctl* call (see below). 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 are accepted. *strlog* accepts 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.

At most one trace logger and one error logger can be active at a time. Once the logger process has identified itself through the *ioctl* call, *log* begins sending up messages subject to the restrictions noted above. These messages are obtained through 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, and a sequence number. 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.

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 and flags 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.

Attempting to issue an I_TRCLOG or I_ERRLOG when a logging process of the given type already exists results in the error ENXIO being returned. Similarly, ENXIO is returned for I_TRCLOG *ioctls* 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).

EXAMPLES

Example of I_ERRLOG notification.

```
struct strioctl ioc;

ioc.ic_cmd = I_ERRLOG;
ioc.ic_timout = 0;        /* default timeout (15 secs.) */
```

```
loc.ic_len = 0;
loc.ic_dp = NULL;

ioctl(log, I_STR, &loc);
```

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 is allowed */
tid[1].ti_level = -1;        /* any level is allowed */

loc.ic_cmd = I_TRCLOG;
loc.ic_timout = 0;
loc.ic_len = 2 * sizeof(struct trace_ids);
loc.ic_dp = (char *)tid;

ioctl(log, I_STR, &loc);
```

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
<sys/log.h>
<sys/strlog.h>

SEE ALSO

strace(1M), strerr(1M), clone(7), intro(2), getmsg(2), putmsg(2).
UNIX System V Release 3.2 Streams Programmer's Guide.

NAME

lp - parallel printer interface

DESCRIPTION

The *lp* driver provides an interface to the parallel printer channel. Bytes written are sent to the printer. Opening and closing produce page ejects. Unlike the serial interfaces [*termio*(7)], the *lp* driver never prepends a carriage return to a newline (line feed). The *lp* driver does have options to filter output for the benefit of printers with special requirements. The driver also controls page format. Page format and filter options are controlled with *ioctl*(2):

```
#include <sys/lprio.h>
ioctl(fildes, command, arg)
```

where *command* is one of the following constants:

LPRGET Get the current page format and put it in the **lprio** structure pointed to by *arg*.

LPRSET Set the current page format from the location pointed to by *arg*; this location is a structure of type **lprio**, declared in the header file:

```
struct lprio {
        short ind;
        short col;
        short line;
};
```

arg should be declared as follows:

```
struct lprio *arg;
```

ind is the page indent in columns, initially 4. *col* is the number of columns in a line, initially 132, *line* is the number of lines on a page, initially 66. A newline that extends over the end of a page is output as a formfeed. Lines longer than the line length minus the indent are truncated.

LPRSOPTS

Set the filter options from *arg*, which must be of type **int**. *Arg* should be the logical or of one or more of the following constants, defined in the header file:

Constant	*Value*	*Meaning*
LPNOBS	4	No back space. Set this bit if the printer cannot properly interpret backspace characters. The driver uses carriage return to produce equivalent overstriking.
LPRAW	8	Raw output. Set this bit if the driver must not edit output in any way. The driver ignores all other option bits.
LPCAP	16	Capitals. This option supports printers with a "half-ASCII" character set. Lowercase is translated to uppercase. The following special characters are translated: { to (-, } to)-; ` to '-; | to !-; ~ to ^-.
LPNOCR	32	No Carriage Return. This option supports printers that do not respond to a carriage return (character 0D hexadecimal). Carriage returns are changed to newlines. If No Newline is also set, carriage returns are changed to form feeds.
LPNOFF	64	No Form Feed. This option supports printers that do not respond to a form feed (character 0C hexadecimal). Form Feeds are changed to newlines. If No Newline is also set, form feeds are changed to carriage returns.

LPNONL	128	No Newline. This option supports printers that do not respond to a newline (character 0A hexadecimal). Newlines are changed to carriage returns. If No Carriage Return is also set, newlines are changed to form feeds.

Setting all three (No Carriage Return, No New Line, and No Form Feed) has the same effect as setting none of them.

LPRGOPTS Return the current state of the filter options.

Note that once set, options remain intact through a *close*.

FILES

/dev/lp*?*

SEE ALSO

lpr(1), lpset(1).

NAME

mem, kmem - system memory interface

DESCRIPTION

The *mem* special file is an image of the core memory of the CTIX-based processor board. It can be used, for example, to examine, and even to patch the system.

Byte addresses in *mem* are interpreted as memory addresses. References to nonexistent 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 *kmem* is the same as *mem*, except that kernel virtual memory rather than physical memory is accessed.

Accessing 0 to 24 megabytes allows a process to read its own space. 0x7F800000 to 0x80000000 allows a process to read the kernel. Invalid pages cause errors to be returned.

FILES

/dev/mem
/dev/kmem

SEE ALSO

vme(7).

NAME

null - the null file

DESCRIPTION

Data written on a null special file, */dev/null*, is discarded.

Reads from a null special file always return 0 bytes.

FILES

/dev/null

NAME

prf - operating system profiler

DESCRIPTION

The special file */dev/prf* provides access to activity information in the operating system. Writing the file loads 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

SEE ALSO

config(1M), profiler(1M).

NAME

qic - interface for QIC tape

DESCRIPTION

This interface provides access to quarter-inch cartridge (QIC) streaming tape. QIC tape drives are supported only as character devices. There can be only one onboard quarter-inch tape drive (**qic**), assigned major device number 18 by default. The minor number specifies whether the tape device is rewind on close, no rewind on close, or ioctl commands only, as follows:

Starting from bit 0, if bit 2 is not set, the device is rewind on close (**/dev/rmt/c0d0**); if bit 2 is set, the device is no rewind on close (**/dev/rmt/c0d0n**); if bit 3 is set, the device allows only ioctl commands (**/dev/rmt/c0d0c**). (Note that on an S/640, QIC drives can be numbered **d0** to **d7**.)

Tape files are separated by tape marks, also known as EOFs. Closing a file open for writing writes one tape mark; if the device was no-rewind, the tape is left positioned just after the single QIC tape mark. It is not possible to overwrite a tape mark. Writing must begin either at the beginning of the tape or after any previously recorded data.

Each *read* or *write* reads or writes the next physical blocks. *Read/write* sizes must be a multiple of 512. *Read/write* buffers must begin on an even address; this is the same alignment as **short**. Seeks are ignored. Reading a tape mark produces a zero-length *read* and leaves the tape positioned before the mark.

The following commands are supported for QIC tape through *ioctl*(2):

```
#include <sys/tsioctl.h>
#include <sys/scsitape.h>
ioctl (fildes, cmd, arg)
```

where *cmd* is one of the following:

TPIOCTYPE	The return is always TPIOC
TPTYPE	The return is always **q**
TPGETA	Get the current status of the tape controller. *Arg* must be a pointer to a *tpio* structure defined as follows:

```
struct tpio {
        unsigned status;
        short    retries;
        short    under;
};
```

TPSETA	Set *tpio* structure.
TPCMD	Specify a command to the tape controller as specified in *arg*. The following are legal values of *arg*:

SENSE	Perform a read tape status. The result can be read through TPGETA.
TRESET	Reset the tape controller.
TSELECT	Determine whether the unit is selectable.
WFM	Write file mark.
RFM	Read file mark.
TCLRERR	Clear any outstanding errors.
REWIND	Issue a rewind command.
ERASE	Issue an erase tape command.
RETEN	Issue a retension tape command.

FILES

/dev/rmt/c0d0	rewind on close device
/dev/rmt/c0d0n	no rewind on close device
/dev/rmt/c0d0c	ioctl-only device
/dev/rmt/c0d[0-7]	S/640 rewind on close SCSI device
/dev/rmt/c0d[0-7]n	S/640 no rewind on close SCSI device
/dev/rmt/c0d[0-7]c	S/640 ioctl-only SCSI device
/dev/rmt0	linked to /dev/rmt/c0d0
/dev/rmt4	linked to /dev/rmt/c0d0n

SEE ALSO

config(1M), scsimap(1M), tapeset(1M), tsioctl(1), system(4), ipt(7), scsi(7), stape(7).

NOTES

Use the *uconf*(1M) command to set **tpiocype_old** if old *ioctl* calls are required (for backward compatibility).

Not all drivers support all TPCMDs.

WARNING

A nondata error cannot be recovered from except by closing the device.

A QIC tape has no special mark for end of tape, as opposed to end of file.

NAME

scsi - scsi control device

SYNOPSIS

#include <sys/scsi.h>
#include <sys/scsiioctl.h>

DESCRIPTION

The special file **/dev/scsi** is an interface to the low-level SCSI driver. This low-level driver is used by all high-level SCSI devices, such as SCSI tape and SCSI disk. This means that there is only one SCSI driver, so all SCSI disk **/dev/dsk** nodes use the same controller number and all SCSI tape **/dev/rmt** nodes use the same controller number, even though the disk drives and the tape drives are liable to be on various busses.

The low-level SCSI driver performs such functions as SCSI bus protocol, SCSI device mapping (logical-to-physical), and SCSI target options (parity, reselect).

All /dev/scsi operations are accessed through ioctl calls.

```
/*
 * ioctl control packets
 */
/* This structure defines all gd<->scsi and tape<->scsi
* mappings */
struct scsiioctl_map {
    unchar  type;    /* SCSI_GDTYPE or SCSI_SATYPE */
    unchar  dev;     /* slot number for gd or tape mapping */
    unchar  flag;    /* 1 = valid entry, 0 = invalid entry */

    unchar  bus;     /* target scsi bus */
    unchar  lun;     /* target lun */
    unchar  target;  /* target controller id */
    unchar  config;  /* configuration bits (parity, disconnect) */
};

/* The following is a template - many map entries should actually
* be supplied */
/* Calling GETMAP with a size of zero is useful for getting the
* "total" value */
```

```
struct scsiioctl_maps {
    ushort  size;     /* The number of maps in the "maps" array */
    ushort  total;    /* The number of maps that the kernel has */
    struct  scsiioctl_map maps[1];
};

#define SCSI_IOCTYPE  0xff00        /* same as TTY IOCTYPE;
                                       identifies type as SCSI;
                                       returns SCSI_IOC */
#define SCSI_IOC      ('s'<<8)
#define SCSI_GETMAP   (SCSI_IOC|3)/* get all valid map entries */
#define SCSI_SETMAP   (SCSI_IOC|4)/* set map */

/* Note:  these device types do not correspond to the SCSI device
 * types:  the intention here is to indicate which device driver
 * should be used as the handler, not the actual device type.
 */
#define SCSI_GDTYPE   0x11    /* General disk devices */
#define SCSI_SATYPE   0x12    /* sequential access devices */
```

SEE ALSO

scsimap (1M).

NAME

stape - SCSI quarter-inch and half-inch tape

DESCRIPTION

The *stape* tape interface provides access to quarter- and half-inch tape drives. By default, the major number for both SCSI tape drives is 65. The interface is the same for both tapes; you specify quarter-inch or half-inch as an option in the !SCSIMAP section of the **/etc/system** (system description) file; see *system*(4). See *tapeset*(1M) for half-inch SCSI tape drive initialization information.

Bits 0 through 7 of the minor device number specify the tape density, rewind option, and whether it is ioctl only, as follows (where "not 4" indicates that bit 4 is *not* set, and "4" indicates that bit 4 *is* set, and so on for each bit):

0 - 2	drive number
3	unused
not 4 and not 5	medium density
4 and not 5	low density
not 4 and 5	high density
4 and 5	use last density
6	rewind on close
7	ioctl only

The following commands are supported for SCSI tape through *ioctl*(2):

```
#include <sys/tsioctl.h>
#include <sys/scsitape.h>
ioctl (fildes, cmd, arg)
```

where *cmd* is one of the following:

TPIOCTYPE	The return is always TPIOC; *arg* should be 0.
TPTYPE	The return is always s; *arg* should be 0.

TPPASSTHRU *Arg* must be a pointer to the **sa_ioctl** structure, defined as follows:

```
struct sa_ioctl {
        uint      command;
        caddr_t   address;
        uint      length;
};
```

where the command field in the structure is either **SA_MODSENSE** or **SA_MODSELECT**.

TPCMD Send a command (*arg*) to the tape controller; *arg* can be one of the following:

SENSE	Perform a read tape status. The result can be read through TPGETA.
TRESET	Reset the tape controller.
TSELECT	Determine whether the unit is selectable.
WFM	Write file mark.
RFM	Read file mark.
REWIND	Issue a rewind command.
ERASE	Issue an erase tape command.
RETENT	Issue a retension tape command.
TCLRERR	Clear any outstanding errors.

FILES

/dev/rmt/c*x*d*x*[0-7]	rewind-on-close device
/dev/rmt/c*x*d*x*[0-7]n	no-rewind-on-close device
/dev/rmt/c*x*d*x*[0-7]c	ioctl-only device

SEE ALSO

config(1M), scsimap(1M), tapeset(1M), tsioctl(1), system(4), ipt(7), qic(7), scsi(7).

NAME

streamio - STREAMS ioctl commands

SYNOPSIS

#include <stropts.h>
int ioctl (fildes, command, arg)
int fildes, command;

DESCRIPTION

STREAMS [see *intro*(2)] ioctl commands are a subset of *ioctl*(2) system calls that perform a variety of control functions on *streams*. The arguments *command* and *arg* are passed to the file designated by *fildes*, and are interpreted by the *stream head*. Certain combinations of these arguments can be passed to a module or driver in the stream.

The *fildes* argument is an open file descriptor that refers to a stream. The *command* argument determines the control function to be performed as described below. The *arg* argument represents additional information needed by this command. The type of *arg* depends on the command, but it is generally an integer or a pointer to a *command*-specific data structure.

Since these STREAMS commands are a subset of *ioctl*, they are subject to the errors described there. In addition to those errors, the call fails 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. 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] The *arg* argument 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*. In an I_POP request, *arg* should be 0. 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. An "#include <sys/conf.h>" declaration is required. On failure, *errno* is set to one of the following values:

[EFAULT]	The *arg* argument 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.

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_SETSIG Informs the stream head that the user wants the kernel to issue the SIGPOLL signal [see *signal*(2) and *sigset*(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 of the following constants:

S_INPUT — A non-priority message has arrived on a stream head read queue, and no other messages existed on that queue before this message was placed there. This is set even if the message is of zero length.

S_HIPRI — A 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_MSG — A STREAMS signal message that contains the SIGPOLL signal has reached the front of the stream head read queue.

A user process can choose to be signaled only of priority messages by setting the *arg* bitmask to the value S_HIPRI.

Processes that should 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 is signaled when the event occurs.

If the value of *arg* is zero, the calling process is unregistered and does not receive further SIGPOLL signals. On failure, *errno* is set to one of the following values:

[EINVAL] — The value of *arg* is invalid or zero, and the 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. *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. If the user sets *flags* to RS_HIPRI, I_PEEK only looks for a priority 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, or if the RS_HIPRI flag was set in *flags* and a priority message was not present 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 0 or RS_HIPRI. 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

I_SRDOPT Sets the read mode using the value of the argument *arg*. Legal *arg* values are:

RNORM	Byte-stream mode, the default.
RMSGD	Message-discard mode.
RMSGN	Message-nondiscard mode.

Read modes are described in *read*(2). On failure, *errno* is set to the following value:

[EINVAL] *arg* is not one of the above legal values.

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. The *fildes* argument 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 stores a pointer. This pointer is the address of the read queue structure of the driver for the stream corresponding to *fildes* in the *strfdinsert* 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. A non-priority message is created if *flags* is set to 0, and a priority message is created if *flags* is set to RS_HIPRI. For non-priority messages, I_FDINSERT blocks if the stream write queue is full due to internal flow control conditions. For priority messages, I_FDINSERT does not block on this condition. For non-priority messages, I_FDINSERT does not block when the write queue is full and O_NDELAY 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 in the stream, regardless of priority or whether O_NDELAY 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 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 *strfdinsert* structure. In this case, *errno* is 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 returns 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 block until the active I_STR completes at the stream head. The default timeout interval for these requests is 15 seconds. The O_NDELAY [see *open*(2)] flag has no effect on this call.

To send requests downstream, *arg* must point to a *strioctl* structure which contains the following members:

```
int     ic_cmd; /* downstream command */
int     ic_timout; /* ACK/NAK timeout */
```

```
int     ic_len;  /* length of data arg */
char    *ic_dp;  /* ptr to data arg */
```

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 waits for acknowledgement before timing out. *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 converts the information pointed to by the *strioctl* structure to an internal ioctl command message and sends 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] 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 fails with *errno* set to the value in the message.

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 integer 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. The *arg* argument is a pointer to a data buffer large enough to hold an *strrecvfd* data structure containing the following members:

```
int fd;
unsigned short uid;
unsigned short 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 is not set [see *open*(2)], I_RECVFD blocks until a message is present at the stream head. If O_NDELAY is set, I_RECVFD fails 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 was not present at the stream head read queue, and the O_NDELAY flag is set.

[EBADMSG] The message at the stream head read queue was 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*.

The following two 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 stream head is linked into a multiplexing configuration in more than one place.

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 fails 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. The *fildes* argument must correspond to the stream on which the *ioctl* I_LINK command was issued to link the stream below 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] The *arg* argument is an invalid multiplexor ID number or *fildes* is not the stream on which the I_LINK that returned *arg* was performed.

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 fails 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), sigset(2), write(2).
UNIX System V Release 3.2 Streams Programmer's Guide.
UNIX System V Release 3.2 Streams Primer.

DIAGNOSTICS

Unless specified otherwise above, the return value from *ioctl* is 0 upon success and -1 upon failure with *errno* set as indicated.

NAME

sxt - STREAMS multiplexor

DESCRIPTION

The special file */dev/sxt* is a streams multiplexor that can be used to multiplex groups of processes over any lower streams. Typically, it is used by the shell layer program [shl(1)] to multiplex terminal I/O.

Sxt devices are named by inodes in the directory */dev/sxt*, and are allocated in groups of eight. To allocate a group, the user process should try to open a file with a name of the form **/dev/sxt/*??*0**, with the FEXCL flag set in the *open* system call until the *open* returns successfully. The last three bits of the minor device number determine the channel number of the *sxt* device. Initially, channel 0 is the control device, until it is switched. Bits 3 to 15 are used as the group number.

Once the *sxt* device is opened, it should be linked with a lower stream, for example, a tty device with a line discipline, to initiate the multiplexing. For example:

```
tty = open("/dev/tty",2);
sxt = open("/dev/sxt000",2);
ioctl(sxt,I_LINK,tty);
```

Channel 0 is always the controlling device, and only the controlling device can issue SXTIOCSWTCH, SXTIOCBLK, SXTIOCUBLK, and SXTIOCSTAT ioctl commands. However, any channel can become the active channel, which is the only channel that can receive messages from lower streams; other channels attempting to read are blocked. Initially, channel 0 is the active channel.

The *sxt* driver supports the following ioctl commands:

I_LINK Link the lower streams to the *sxt* device.

I_UNLINK
: Unlink the multiplexor. This is done automtically on close.

SXTIOCSWTCH
: Switch to the channel specified by the argument in the ioctl call.

SXTIOCWF
: Wait until the device becomes active. The controlling channel becomes active on receipt of a line switch message from lower streams. In the current implementation, the line switch message is of the type M_CTL with the first character in the data block equal to **Z**. The default line discipline generates this message on receipt of the line switch character (default Control-z) from the keyboard.

SXTIOCBLK
: Block output for the channel.

SXTIOCUBLK
: Unblock pending output for the channel.

SXTIOCSTAT
: Get the status (blocked on input or output) of each channel and store in the *sxtblock* structure referenced by the argument.

Any other ioctl commands that are not understood by the *sxt* driver are passed downstream.

When the controlling channel is closed, all other channels are closed, and a hangup control message is sent to the queue heads.

Refer to *streamio*(7) for possible return values from I_LINK and I_UNLINK. Return values for the other commands follow:

EPERM	Command not executed from channel 0.
EINVAL	Argument is out of range.
ENXIO	The channel to switch to has not been opened.
EAGAIN	No streams buffers to process the request.

FILES

/dev/sxt/???

SEE ALSO

shl(1), stty(1), ioctl(2), open(2), streamio(7), termio(7).
CTIX Network Programmer's Primer.
UNIX System V Release 3.2 Network Programmer's Guide.
UNIX System V Release 3.2 Streams Programmer's Guide.
UNIX System V Release 3.2 Streams Primer.

BUGS

The */dev/sxt* driver works only with STREAMS devices.

NAME

tcp - Internet Transmission Control Protocol

SYNOPSIS

#include <sys/socket.h>
#include <sys/in.h>

s = socket(AF_INET, SOCK_STREAM, 0);

DESCRIPTION

The TCP protocol provides reliable, flow-controlled, two-way transmission of data. It is a byte-stream protocol used to support the SOCK_STREAM abstraction. TCP uses the standard Internet address format and, in addition, provides a per-host collection of "port addresses." Thus, each address is composed of an Internet address specifying the host and network, with a specific TCP port on the host identifying the peer entity.

Sockets using the *tcp* protocol are either "active" or "passive." Active sockets initiate connections to passive sockets. By default TCP sockets are created active; to create a passive socket the *listen*(2) system call must be used after binding the socket with the *bind*(2) system call. Only passive sockets can use the *accept*(2) call to accept incoming connections. Only active sockets can use the *connect*(2) call to initiate connections.

Passive sockets may "underspecify" their location to match incoming connection requests from multiple networks. This technique, called "wildcard addressing," allows a single server to provide service to clients on multiple networks. To create a socket that listens on all networks, the Internet address INADDR_ANY must be bound. The TCP port can still be specified at this time; if the port is not specified the system will assign one. Once a connection has been established the socket's address is fixed by the peer entity's location. The address assigned the socket is the address associated with the network interface through which packets are being transmitted and received. Normally this address corresponds to the peer entity's network.

TCP supports one socket option which is set with *setsockopt* and tested with *getsockopt* [see *getsockopt*(2)]. 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 acknowledgment 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 (from **<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*(3)].

Options at the IP transport level can be used with TCP; see *ip*(7). Incoming connection requests that are source-routed are noted, and the reverse source route is used in responding.

TCP is also available as a TLI connection-oriented protocol via the special files **/dev/inet/tcpord** and **/dev/int/tcpdis.** The **tcpord** device supports Orderly Release. If the **tcpdis** device is used, any remote disconnect will be interpreted as a Disconnect Request. TCP options are supported via the TLI options mechanism.

FILES

/dev/inet/tcpord, /dev/inet/tcpdis

SEE ALSO

getsockopt(2), socket(2), intro(7), inet(7), ip(7).
CTIX Network Administrator's Guide.

DIAGNOSTICS

A socket operation may fail with one of the following errors returned:

[EISCONN]	when trying to establish a connection on a socket which already has one
[ENOSR]	when the system runs out of memory for an internal data structure
[ETIMEDOUT]	when a connection was dropped due to excessive retransmissions
[ECONNRESET]	when the remote peer forces the connection to be closed
[ECONNREFUSED]	when the remote peer actively refuses connection establishment (usually because no process is listening to the port)
[EADDRINUSE]	when an attempt is made to create a socket with a port which has already been allocated
[EADDRNOTAVAIL]	when an attempt is made to create a socket with a network address for which no network interface exists

NAME

termio - general terminal interface

DESCRIPTION

CTIX systems use a single interface convention for all RS-232 and cluster (RS-422) terminals, although cluster terminals do not use all the features of the convention. The convention is almost completely taken from the UNIX System V interface for asynchronous terminals.

Two kinds of terminals use this convention:

- RS-232 terminals connected to channels on the computer itself.
- Cluster terminals. Generally a cluster channel supports more than one terminal and some terminals are indirectly connected through other (daisy-chained) terminals. Cluster terminals use the same interface as directly connected RS-232 terminals, except that hardware control operations are meaningless on cluster terminals. (Note that ''cluster terminal'' refers to the way the terminal is used, not to the terminal itself; a Convergent Technologies PT (or GT) terminal can serve as an RS-232 terminal or as a cluster terminal.)

A single naming convention applies to regular RS-232 and cluster terminals. A direct RS-232 or cluster terminal has a name of the form **/dev/tty***xxx*, where *xxx* is the terminal's number expressed in three digits.

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 *getty* and become a user's standard input, output, and error files. The very first terminal file opened by the process group leader of a terminal file not already associated with a process group becomes the *control terminal* for that process group. The control terminal plays a special role in handling quit and interrupt signals, as discussed below. The control terminal is inherited by a child process during a *fork*(2). A process can break this association by changing its process group using *setpgrp*(2).

A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters can be typed at any time, even while output is occurring, and are lost only when the system's character input buffers become completely full, which is rare, or when the user has accumulated the maximum allowed number of input characters that have not yet been read by some program. Currently, this limit is 256 characters. When the input limit is reached, the buffer is flushed and all the saved characters are thrown away without notice.

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 is suspended until an entire line has been typed. Also, no matter how many characters are requested in the read call, at most one line is returned. However, it is not necessary to read a whole line at once; any number of characters can be requested in a read, even one, without losing information.

During input, erase and kill processing is normally done. By default, Control-H (ASCII BS) erases the last character typed, except that it does not erase beyond the beginning of the line. By default, the character @ kills (deletes) the entire input line, and optionally outputs a newline character. Both characters operate on a keystroke basis, independent of any backspacing or tabbing that may have been done. Both the erase and kill characters can be entered literally by preceding them with the escape character (\). In this case, the escape character is not read. The erase and kill characters can be changed.

Certain characters have special functions on input. These functions and their default character values are summarized as follows:

INTR (Rubout or ASCII DEL) Generates an *interrupt* signal, which is sent to all processes with the associated control terminal. Normally, each such process is forced to terminate, but arrangements can be made to ignore the signal or to receive a trap to an agreed-upon location; see *signal*(2).

QUIT (Control- | or ASCII FS) Generates a *quit* signal. Its treatment is identical to the interrupt signal except that, unless a receiving process has made other arrangements, it is not only terminated, but a core image file (called **core**) is created in the current working directory.

SWTCH (Control-Z or ASCII SUB) Used by the job control facility, *shl*, to change the current layer to the control layer.

ERASE (Control-H or ASCII BS) Erases the preceding character. It does not erase beyond the start of a line, as delimited by an NL, EOF, or EOL character.

KILL (@) Deletes the entire line, as delimited by an NL, EOF, or EOL character.

EOF (Control-D or ASCII EOT) Can be used to generate an end-of-file from a terminal. When received, all characters waiting to be read are immediately passed to the program, without waiting for a newline, and the EOF is discarded. Thus, if there are no characters waiting, which is to say the EOF occurred at the beginning of a line, zero characters are passed back, which is the standard end-of-file indication.

NL (ASCII LF) The normal line delimiter. It can not be changed or escaped.

EOL (ASCII NUL) An additional line delimiter, like NL. It is not normally used.

EOL2 Another additional line delimiter.

STOP (Control-S or ASCII DC3) Can be used to temporarily suspend output. 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 (Control-Q or ASCII DC1) Used to resume output which has been suspended by a STOP character. While output is not suspended, START characters are ignored and not read. The start/stop characters can not be changed or escaped.

The character values for INTR, QUIT, SWTCH, ERASE, KILL, EOF, and EOL can be changed to suit individual tastes. The ERASE, KILL, and EOF characters may be escaped by a preceding \ character, in which case no special function is done.

When the carrier signal from the data-set drops, a *hang-up* signal is sent to all processes that have this terminal as the control terminal. Unless other arrangements have been made, this signal causes the processes to terminate. If the hang-up signal is ignored, any subsequent read returns with an end-of-file indication. Thus, programs that read a terminal and test for end-of-file can terminate appropriately when hung up on.

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 by putting them in the output queue as they arrive. If a process produces characters more rapidly than they can be typed, it is suspended when its output queue exceeds some limit. When the queue has drained down to some threshold, the program is resumed.

Several *ioctl*(2) system calls apply to terminal files. The primary calls use the following structure, defined in **<termio.h>**:

```
#define  NCC       8
struct   termio {
         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 relative positions and initial values for each function are as follows:

0	VINTR	DEL
1	VQUIT	FS
2	VERASE	BS
3	VKILL	@
4	VEOF	EOT
5	VEOL	NUL
6	reserved	NUL
7	VSWTCH	NUL

The *c_iflag* field describes the basic terminal input control:

IGNBRK	0000001	Ignore break condition.
BRKINT	0000002	Signal interrupt on break.
IGNPAR	0000004	Ignore characters with parity errors.
PARMRK	0000010	Mark parity errors.
INPCK	0000020	Enable input parity check.
ISTRIP	0000040	Strip character.
INLCR	0000100	Map NL to CR on input.
IGNCR	0000200	Ignore CR.
ICRNL	0000400	Map CR to NL on input.
IUCLC	0001000	Map upper-case to lower-case on input.
IXON	0002000	Enable start/stop output control.
IXANY	0004000	Enable any character to restart output.
IXOFF	0010000	Enable start/stop input control.

If IGNBRK is set, the break condition (a character framing error with data all zeros) is ignored, that is, not put on the input queue and therefore not read by any process. Otherwise, if BRKINT is set, the break condition generates an interrupt signal and flush both the input and output queues. If IGNPAR is set, characters with other framing and parity errors are ignored.

If PARMRK is set, a character with a framing or parity error which is not ignored is read as the three-character sequence: 0377, 0, X, where X is the data of the character received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid character of 0377 is read as 0377, 0377. If PARMRK is not set, a framing or parity error which is not ignored is read as the character NUL (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.

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. All start/stop characters are ignored and not read. If IXANY is set, any input character restarts output which has been suspended.

If IXOFF is set, the system transmits START/STOP characters when the input queue is nearly empty/full.

The initial input control value is BRKINT, IGNPAR, ISTRIP, ICRNL, IXOFF, IXON.

The *c_oflag* field specifies the system treatment of output:

OPOST	0000001	Postprocess output.
OLCUC	0000002	Map lower case to upper on output.
ONLCR	0000004	Map NL to CR-NL on output.
OCRNL	0000010	Map CR to NL on output.
ONOCR	0000020	No CR output at column 0.
ONLRET	0000040	NL performs CR function.
OFILL	0000100	Use fill characters for delay.
OFDEL	0000200	Fill is DEL, else NUL.
NLDLY	0000400	Select newline delays:
NL0	0	
NL1	0000400	
CRDLY	0003000	Select carriage return delays:
CR0	0	
CR1	0001000	
CR2	0002000	
CR3	0003000	
TABDLY	0014000	Select horizontal-tab delays:

TAB0	0	
TAB1	0004000	
TAB2	0010000	
TAB3	0014000	Expand tabs to spaces.
BSDLY	0020000	Select backspace delays:
BS0	0	
BS1	0020000	
VTDLY	0040000	Select vertical-tab delays:
VT0	0	
VT1	0040000	
FFDLY	0100000	Select form-feed delays:
FF0	0	
FF1	0100000	

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 ONLRET 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 is 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 which need only a minimal delay. If OFDEL is set, the fill character is DEL; otherwise, NUL.

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, 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.

The *c_cflag* field describes the hardware control of the terminal:

CBAUD	0000017	Baud rate:
B0	0	Hang up
B50	0000001	50 baud
B75	0000002	75 baud
B110	0000003	110 baud
B134	0000004	134 baud
B150	0000005	150 baud
B200	0000006	200 baud
B300	0000007	300 baud
B600	0000010	600 baud
B1200	0000011	1200 baud
B1800	0000012	1800 baud
B2400	0000013	2400 baud
B4800	0000014	4800 baud
B9600	0000015	9600 baud
B19200	0000016	19200 baud
EXTA	0000016	External A
B38400	0000017	38400 baud
EXTB	0000017	External B
CSIZE	0000060	Character size:
CS5	0	5 bits
CS6	0000020	6 bits
CS7	0000040	7 bits
CS8	0000060	8 bits
CSTOPB	0000100	Send two stop bits, else one.
CREAD	0000200	Enable receiver.
PARENB	0000400	Parity enable.
PARODD	0001000	Odd parity, else even.

HUPCL	0002000	Hang up on last close.
CLOCAL	0004000	Local line, else dial-up.
LOBLK	0010000	Block layer output.
CTSCD	0020000	Use hardware flow control.

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. 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. 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.

If LOBLK is set, the output of a job control layer is blocked when it is not the current layer. Otherwise, the output generated by that layer is multiplexed onto the current layer.

If CTSCD is set, flow control is performed using hardware signals. No data is sent in the absence of CTS (Clear to Send) signal. Outgoing data is suspended if CTS is lowered; transmission is resumed after CTS is raised.

The initial hardware control value after open is B9600, CS8, CREAD, HUPCL.

The *c_lflag* field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline (0) provides the following:

ISIG	0000001	Enable signals.
ICANON	0000002	Canonical input (erase and kill processing).
XCASE	0000004	Canonical upper/lower presentation.
ECHO	0000010	Enable echo.

ECHOE	0000020	Echo erase character as BS-SP-BS.
ECHOK	0000040	Echo NL after kill character.
ECHONL	0000100	Echo NL.
NOFLSH	0000200	Disable flush after interrupt or quit.

If ISIG is set, each input character is checked against the special control characters INTR, SWTCH, and QUIT. 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. These functions may be disabled individually by changing the value of the control character to an unlikely or impossible value (for example, 0377).

If ICANON is set, canonical processing is enabled. For a STREAMS tty driver (the default for all RS-232 ports), when ICANON is set, a read returns all characters on the buffer, up to the first delimiter (such as newline). The STREAMS tty driver allocates 256 bytes for canonical processing: if more than 256 characters are read before a delimiter occurs, the rest are truncated.

For STREAMS and non-STREAMS tty drivers, setting ICANON enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, and EOL. 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 MIN and TIME values are stored in the position for the EOF and EOL characters, respectively. The time value represents tenths of seconds. The values of VMIN and VTIME control how many and when characters are returned. If both are 0, reads come back immediately if no characters are present. If VMIN is greater than 0 and VTIME is equal to 0, the read waits until at least VMIN characters have been received. If VMIN is equal to 0 and VTIME is greater than 0, the read returns after VTIME tenths of a second, regardless of whether any characters have been received. Note that in this case a read may return 0, which is indistinguishable from end-of-file. If VMIN is greater than 0 and VTIME is greater than 0, the timeout period starts after the first character has been received; thus, a read always returns greater than or equal to 1.

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. If ECHO and ECHOE are set, the erase character is echoed as ASCII BS SP BS, which clears the last character from a CRT screen. If ECHOE is set and ECHO is not set, the erase character is echoed as ASCII SP BS. If ECHOK is set, the NL character is echoed after the kill character to emphasize that the line is deleted. Note that an escape character preceding the erase or kill character removes any special function. 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). Unless escaped, the EOF character is not echoed. Because EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.

If NOFLSH is set, the normal flush of the input and output queues associated with the quit, switch, and interrupt characters is not performed.

The initial local modes are as follows: ISIG, ICANON, ECHO, ECHOK .

The initial line-discipline is 0.

The primary *ioctl*(2) system calls have the following form:

```
ioctl (fildes, command, arg)
struct termio *arg;
```

The commands using this form are as follows:

TCGETA Get the parameters associated with the terminal and store in the *termio* structure referenced by **arg**.

TCSETA Set the parameters associated with the terminal from the structure referenced by **arg**. The change is immediate.

TCSETAW Wait for the output to drain before setting the new parameters. This form should be used when changing parameters that affect output.

TCSETAF Wait for the output to drain, then flush the input queue and set the new parameters.

Additional *ioctl*(2) calls have the following form:

```
ioctl (fildes, command, arg)
int arg;
```

The commands using this form are as follows:

TCSBRK Wait for the output to drain. If *arg* is 0, then send a break (zero bits for 0.25 seconds).

TCXONC Start/stop control. If *arg* is 0, suspend output; if 1, restart suspended output; if 2, transmit XOFF; if 3, transmit XON.

TCFLSH If *arg* is 0, flush the input queue; if 1, flush the output queue; if 2, flush both the input and output queues.

TCGEXT Get the parameters associated with the terminal. The parameters are passed back as the return value from the *ioctl*(2) function. The return value is defined as follows (see **/usr/include/sys/tty.h**):

```
#define CDBIT   0x04 /* TCGEXT: CD is present  */
#define CTSBIT  0x08 /* TCGEXT: CTS is present */
#define DSRBIT  0x10 /* TCGEXT: DSR is present */
#define RIBIT   0x20 /* TCGEXT: RI is present  */
```

TCSEXT Set the parameters associated with the terminal from arg. The bits in arg are defined as follows:

```
#define RTSBIT 0x08 /* TCSEXT: set/clear RTS  */
#define DTRBIT 0x10 /* TCSEXT: set/clear DTR  */
#define SETEXT 0x80 /* TCSEXT: 1 = set, 0 = clear */
```

TCSEXTW
Wait for the output to drain before setting the new parameters as in TCSEXT.

The TCGEXT command provides the status of CD, CTS, DSR, and RI. The TCSEXT and TCSEXTW commands allow setting or clearing of the RTS and DTR signals. If the SETEXT bit is set, the RTS and/or DTR lines can be turned on. If the SETEXT bit is cleared, either or both of these lines can be turned off.

FILES

/dev/tty*
/dev/tp*

SEE ALSO

stty(1), fork(2), ioctl(2), setpgrp(2), signal(2), tp(7), tty(7).

WARNING

The default value for ERASE is backspace rather than the historical #.

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 can be consumed by a transport protocol provider that supports the Transport Interface. This allows a user to initiate certain TI functions as atomic operations.

The *timod* module must be pushed (see the *UNIX System V Release 3.2 Streams Primer*) onto only a *stream* terminated by a transport protocol provider that 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).
UNIX System V Release 3.2 Streams Programmer's Guide.
UNIX System V Release 3.2 Streams Primer.
UNIX System V Release 3.2 Network Programmer's Guide.

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

tiop - terminal accelerator interface

SYNOPSIS

#include <sys/tiop.h>

DESCRIPTION

The *tiop* driver provides loading and unloading functions for the terminal accelerator. The open of device **/dev/tiop** fails if a terminal accelerator board is not present or if the board is already loaded. The only valid function after opening the *tiop* device is an *ioctl* call to download the accelerator. The following command is supported through *ioctl:*

IOPATTACH — Download the IOP; Note that *arg* must point to an area in the caller's space where the first four bytes are a count of the number of bytes to be loaded into the accelerator. The actual data must follow the count field immediately. The count bytes are copied into the accelerator starting at memory location 0. After loading, the accelerator is reset and begins execution at 0 in its memory. After a successful IOPATTACH, all but two onboard RS-232 ports are controlled by the accelerator.

NAME

tirdwr - Transport Interface read/write interface STREAMS module

DESCRIPTION

tirdwr is a STREAMS module that provides an alternate interface to a transport provider (such as TCP) which supports the Transport Interface (TI) functions of the Network Services library. 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 can 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

intro(2), getmsg(2), putmsg(2), read(2), write(2), intro(3), streamio(7), timod(7).
UNIX System V Release 3.2 Streams Programmer's Guide.
UNIX System V Release 3.2 Streams Primer.
UNIX System V Release 3.2 Network Programmer's Guide.

NAME

tp - controlling terminal's local RS-232 channels

DESCRIPTION

The **tp** device accesses the RS-232 channels on the controlling terminal. The terminal must be a cluster terminal configured to permit use of the local RS-232 channels [see *termio*(7)]. Just as **/dev/tty** permits a process to conveniently access its process group's controlling terminal [(see *tty*(7)], **/dev/tpa** and **/dev/tpb** access the controlling terminal's RS-232 channels without reference to the terminal number. This is convenient for accessing the user's local hardware, such as a telephone with an RS-232 interface.

SEE ALSO

tty(7).

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 should 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.

If the terminal is under window management, a process group is controlled by a specific window and I/O on */dev/tty* is directed to that window.

A terminal can control one process group in each window. See *window*(7).

All RS-232 ports are controlled by the STREAMS tty driver, which handles buffering as follows: the line discipline module allocates a 4-byte STREAMS buffer to perform echoing, a 256-byte STREAMS buffer for input processing, and a variable-size (from 4 to 512 bytes) STREAMS buffer for output processing.

FILES

/dev/tty
/dev/tty*

SEE ALSO

termio(7), tp(7), window(7).

NAME

udp - Internet User Datagram Protocol

SYNOPSIS

#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_DGRAM, 0);

DESCRIPTION

UDP is a simple, unreliable datagram protocol that is used to support the SOCK_DGRAM abstraction for the Internet protocol family. UDP sockets are connectionless and are normally used with the *sendto* and *recvfrom* calls; the *connect*(2) call can also be used to fix the destination for future packets (in which case the *recv*(2) or *read*(2) and *send*(2) or *write(2)* system calls can be used). In addition, UDP is available as TLI connectionless transport via the special file **/dev/inet/udp**.

UDP address formats are identical to those used by TCP. In particular, UDP provides a port identifier in addition to the normal Internet address format. Note that the UDP port space is separate from the TCP port space (that is, a UDP port cannot be ''connected'' to a TCP port). In addition, broadcast packets can be sent (assuming the underlying network supports this) by using a reserved broadcast address; this address is network interface dependent.

Options at the IP transport level can be used with UDP; see *ip*(7).

FILES

/dev/inet/udp

SEE ALSO

getsockopt(2), recv(2), send(2), socket(2), intro(7), inet(7), ip(7).
CTIX Network Administrator's Guide.

DIAGNOSTICS

A socket operation may fail with one of the following errors returned:

[EISCONN] when trying to establish a connection on a socket that already has one, or when trying to send a datagram with the destination address specified and the socket is already connected

[ENOTCONN] when trying to send a datagram, but no destination address is specified, and the socket hasn't been connected

[ENOSR] when the system runs out of memory for an internal data structure

[EADDRINUSE] when an attempt is made to create a socket with a port that has already been allocated

[EADDRNOTAVAIL]
when an attempt is made to create a socket with a network address for which no network interface exists

NAME

vme - VME bus interface

DESCRIPTION

vme files are a set of special files that are images of the VME bus. They can be used, for example, to examine and modify memory and registers on the VME bus.

Byte addresses in *vme* are interpreted as memory addresses. For a read, references to nonexistent locations cause errors to be returned; for a write, nothing is written and no error is returned.

Examining and patching device registers is likely to cause unexpected results when read-only or write-only bits are present.

The structure for *ioctl* calls follows:

```
#define   VMGETREG   ('v'+0)
#define   VMSETREG   ('v'+1)

struct    vmeioctl {
          unchar   vm_mreg;
          unchar   mv_preg;
          unchar   vm_lreg;
};
```

The standard VME interface EEPROM contents follow:

```
#define VME_SLOTS          16
struct   vmeeprom {
    int checksum;          /* Make the entire prom checksum to -1 */
    int flags;             /* EEPROM flags (diag/unix) */
    ushort codeoffset;     /* Offset into EEPROM from the start of code */
    char unused[2];        /* unused, reserved */
    struct {
        char type;         /* Board identification for this slot */
        char unused[7];    /* reserved for future use */
        uint address;      /* Address of the board; in S/MT I/O space */
        uint length;       /* Amount of address space taken up by the board */
        int (*initfp)();   /* Pointer to an optional initialization function */
    } slots[VME_SLOTS];
    char drivers[7860];    /* Reserve the rest for controller code */
};

#define VMEE_DIAG       0  /* Diag has cleared/set EEPROM */
#define VMEE_LOADED     1  /* unix has loaded driver information */
```

```
#define VMET_CMC      1  /* CMC Ethernet controller */
#define VMET_V3200    2  /* Interphase SMD controller */
#define VMET_VTAPE    4  /* Interphase tape controller */
#define VMET_MPCC     5  /* MPCC */
```

FILES

/dev/vme/a16	64K bytes of short address space
/dev/vme/a24	32M bytes of standard address space
/dev/vme/a32l	low 2 gigabytes of extended address space
/dev/vme/a32h	high 2 gigabytes of extended address space
/dev/vme/eeprom	8K VME interface EEPROM

SEE ALSO

ldeeprom(1M), system(4), mem(7).
S/MT Series VME Expansion Technical Reference

NAME

vt - virtual terminal

DESCRIPTION

A virtual terminal provides a terminal-like communication channel between two processes. Each virtual terminal consists of two devices: a slave device, whose name is of the form **/dev/ttyp***xx*, where *xx* is the virtual terminal number; and a master device, whose name is of the form **/dev/vt***xx*, where *xx* is the virtual terminal number. The slave device responds to system calls just like a real terminal [see *termio*(7)] so that it can control interactive programs such as *vi*. But instead of doing actual input/output, reads and writes on the slave device are written and read on the corresponding master device by another process. A typical use of a virtual terminal is to put a network server on the master device and login program on the slave.

The master virtual terminal driver is listed as **ptc** in the **/etc/master** file; the slave virtual terminal driver is listed as **pts**.

The number of virtual terminals must be configured; see *config*(1M) for details.

The process on the master device can exercise flow control on the slave device, much as a real terminal would use XON/XOFF to exercise flow control on a terminal device. The parameterless *ioctl*(2) **TIOCSTOP** stops output to the slave device as if with an **XOFF** character; the parameterless *ioctl*(2) **TIOCSTART** restarts output, as if with an XON character.

FILES

/dev/ttyp??	slave devices
/dev/vt??	master devices
/etc/master	

SEE ALSO

config(1M), ttyname(3C), termio(7).

NAME

window - window management primitives

SYNOPSIS

#include <sys/window.h>

DESCRIPTION

Window management [*wm*(1)] provides a superset of windowless terminal features on a Convergent Technologies Programmable Terminal or Graphics Terminal using an RS-422 connection. This entry describes terminal file features special to window management. Window management features are designed not to interfere with programs that do not know about window management. Such design includes simple extensions to the UNIX system's standard concepts of file descriptor and control terminal.

- Each terminal file descriptor has an associated window number, a small positive integer that identifies a window. A window number is the most primitive way to refer to a window and should not be confused with the window ID used by window management subroutines. A new window gets the smallest window number not already in use. Closing a window frees its number for possible assignment to a later window. Output and control calls on the file descriptor apply only to the descriptor's window; input calls succeed only when the window is active. trIP A file descriptor created by a *dup*(2) or inherited across a *fork*(2) inherits the original descriptor's window number. All the file descriptors in such a chain of inheritance, provided they belong to processes in the same process group, are affected when *ioctl* changes the window number of any of them.
- When a process group's control terminal is under window managment, the process group is actually controlled by a particular window. Such can have more than one process group, each controlled by a different window. Keyboard-generated signals (*interrupt* and *quit*) go to the process group controlled by the active window.

When the user creates a new window by using the SPLIT key, the window manager forks a process for that window. The new process inherits file descriptors for standard input (0), standard output (1), and standard error (2) that are associated with the new window. The new process is leader of a process group controlled by the new window. The new process also inherits the environment of the parent process, which is the window manager itself.

Programs that create and use windows use window management *ioctl*(2) calls. Such calls take the form

```
ioctl (fildes, command, arg)
struct wioctl *arg;
```

fildes is a file descriptor for terminal and window affected, *command* is a window management command (see below) *arg* is a pointer to the following structure, declared in **<sys/window.h>**:

```
#define NWCC 2

struct wioctl {
        wndw_t    wi_dfltwndw;
        wndw_t    wi_wndw;
        slot_t    wi_mycpuslot;
        slot_t    wi_destcpuslot;
        port_t    wi_bport;
        char      wi_dummy;
        unsigned  char wi_cc[NWCC];
};
```

Window management *ioctl* calls get (WIOCGET) and set (WIOCSET and WIOCSETP) terminal attributes described in the *wioctl* structure:

wi_dfltwndw 283u
: The window number for the process's default window. If the process does an *open* on **/dev/tty**, the new file descriptor is associated with the default window.

wi_wndw
: The window number for the window that *fildes* (*ioctl's* first parameter) is associated with.

wi_mycpuslot
: (This field is required for historical reasons; it is not meaningful to the hostprocessor.)

wi_destcpuslot
: (This field is required for historical reasons; it is not meaningful to the host processor.)

wi_bport
: (This field is required for historical reasons; it is not meaningful to the host processor.)

wi_cc (This field is required for historical reasons; it is not meaningful to the host processor.) Not used by the CTIX kernel. A value supplied by a WIOCSET or WIOCSETP is stored in a place associated with window *wp_wndw*. A subsequent WIOCGET on the same window retrieves the information.

The window management *ioctl* commands follow:

WIOCGET 351u
: Get information on calling process and file descriptor *fildes*. Fill in *arg*.

WIOCSET
: Set values for calling process and file descriptor *fildes* from information in *arg*. Has no effect on process group-control terminal relationship.

WIOCSETP
: Set values for calling process and file descriptor *fildes* from information in *arg*. The window specified in *arg->wi_wndw* becomes the process's group's controlling terminal provided the following:

- The calling process is the process group leader.
- The process group is not currently controlled by another window on this or any other terminal.
- The specified window is not already a control window.

WIOCLRP
: Valid only when executed by process group leader. The process group ceases to have a control terminal or window and the control terminal/window ceases to control any process group. The process group is free to find another control terminal/window, and the old control terminal/window is free to become the control terminal/window for another process group.

WIOCCLUSTER
: *ioctl* returns 1 if and only if the terminal is a cluster terminal.

WIOCDIRECT
: Enable direct sending of terminal IPC requests.

WIOCUNDIRECT
: Disable direct sending of terminal IPC requests.

An *open* on a terminal special file other than **/dev/tty** (for example, **/dev/tty256**) produces a file descriptor for the lowest-numbered open window. *ioctl* can move this file descriptor to any window.

An *open* can also obtain a controlling terminal/window. The requirements are the same as for WIOCSETP.

FILES

/dev/tty - control terminal
/dev/tty??? - terminals

SEE ALSO

stty(1), wm(1), dup(2), fork(2), ioctl(2), open(2), wmgetid(3X), wmlayout(3X), wmop(3X), wmsetid(3X), termio(7), tty(7).

WARNINGS

WIOCDIRECT and WIOCUNDIRECT are required by the operating system. Their use by user programs is not recommended.