A/UX® Programmer's Reference

Sections 3(M-Z), 4, and 5

030-0785

Apple Computer, Inc.
20525 Mariani Ave.
Cupertino, California 95014
(408) 996-1010

Simultaneously published in the United States and Canada.

030-0785

Limited Warranty on Media and Replacement

If you discover physical defects in the manual or in the media on which a software product is distributed, Apple will replace the media or manual at no charge to you provided you return the item to be replaced with proof of purchase to Apple or an authorized Apple dealer during the 90-day period after you purchased the software. In addition, Apple will replace damaged software media and manuals for as long as the software product is included in Apple's Media Exchange Program. While not an upgrade or update method, this program offers additional protection for up to two years or more from the date of your original purchase. See your authorized Apple dealer for program coverage and details. In some countries the replacement period may be different, check with your authorized Apple dealer.

ALL IMPLIED WARRANTIES ON THIS MANUAL, INCLUDING IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF THE ORIGINAL RETAIL PURCHASE OF THIS PRODUCT.

Even though Apple has reviewed this manual, **APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS MANUAL, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS MANUAL IS SOLD "AS IS," AND YOU, THE PURCHASER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.**

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS MANUAL, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

030-0785

A/UX Programmer's Reference

Contents

Preface	
Introduction	
Section 3	Subroutines (M-Z)
Section 4	File Formats
Section 5	Miscellaneous Facilities

Preface

Conventions Used in This Manual

A/UX® manuals follow certain conventions regarding presentation of information. Words or terms that require special emphasis appear in specific fonts within the text of the manual. The following sections explain the conventions used in this manual.

Significant fonts

Words that you see on the screen or that you must type exactly as shown appear in `Courier` font. For example, when you begin an A/UX work session, you see the following on the screen:

```
login:
```

The text shows `login:` in `Courier` typeface to indicate that it appears on the screen. If the next step in the manual is

Enter `start`

`start` appears in `Courier` to indicate that you must type in the word. Words that you must replace with a value appropriate to a particular set of circumstances appear in *italics*. Using the example just described, if the next step in the manual is

`login:` *username*

you type in your name—`Laura`, for example— so the screen shows:

```
login: Laura
```

Key presses

Certain keys are identified with names on the keyboard. These modifier and character keys perform functions, often in combination with other keys. In the manuals, the names of these keys appear in the format of an Initial Capital letter followed by SMALL CAPITAL letters.

The list that follows provides the most common keynames.

RETURN	DELETE	SHIFT	ESCAPE
OPTION	CAPS LOCK	CONTROL	

For example, if you enter

`Applee`

instead of

`Apple`

you would position the cursor to the right of the word and press the DELETE key once to erase the additional *e*.

For cases in which you use two or more keys together to perform a specific function, the keynames are shown connected with hyphens. For example, if you see

Press CONTROL-C

you must press CONTROL and C simultaneously (CONTROL-C normally cancels the execution of the current command).

Terminology

In A/UX manuals, a certain term can represent a specific set of actions. For example, the word *Enter* indicates that you type in an entry and press the RETURN key. If you were to see

Enter the following command: `whoami`

you would type `whoami` and press the RETURN key. The system would then respond by identifying your login name.

Here is a list of common terms and their corresponding actions.

Term	Action
Enter	Type in the entry and press the RETURN key
Press	Press a *single* letter or key *without* pressing the RETURN key
Type	Type in the letter or letters *without* pressing the RETURN key
Click	Press and then immediately release the mouse button

Term	Action
Select	Position the pointer on an item and click the mouse button
Drag	Position the pointer on an icon, press and hold down the mouse button while moving the mouse. Release the mouse button when you reach the desired position.
Choose	Activate a command title in the menu bar. While holding down the mouse button, drag the pointer to a command name in the menu and then release the mouse button. An example is to drag the File menu down until the command name Open appears highlighted and then release the mouse button.

Syntax notation

A/UX commands follow a specific order of entry. A typical A/UX command has this form:

`command` [*flag-option*] [*argument*] . . .

The elements of a command have the following meanings.

Element	Description
`command`	Is the command name.
flag-option	Is one or more optional arguments that modify the command. Most flag-options have the form [-opt...] where opt is a letter representing an option. Commands can take one or more options.
argument	Is a modification or specification of the command; usually a filename or symbols representing one or more filenames.

Element	Description
brackets ([ ])	Surround an optional item—that is, an item that you do not need to include for the command to execute.
ellipses (...)	Follow an argument that may be repeated any number of times.

For example, the command to list the contents of a directory (`ls`) is followed below by its possible flag options and the optional argument *names*.

```
ls [-R] [-a] [-d] [-C] [-x] [-m] [-l] [-L]
[-n] [-o] [-g] [-r] [-t] [-u] [-c] [-p] [-F]
[-b] [-q] [-i] [-s] [names]
```

You can enter

```
ls -a /users
```

to list all entries of the directory `/users`, where

`ls`	Represents the command name
`-a`	Indicates that *all* entries of the directory be listed
`/users`	Names which directory is to be listed

Command reference notation

Reference material is organized by section numbers. The standard A/UX cross-reference notation is

cmd(sect)

where *cmd* is the name of the command, file, or other facility; *sect* is the section number where the entry resides.

- ☐ Commands followed by section numbers (1M), (7), or (8) are listed in *A/UX System Administrator's Reference.*
- ☐ Commands followed by section numbers (1), (1C), (1G), (1N), and (6) are listed in *A/UX Command Reference.*
- ☐ Commands followed by section numbers (2), (3), (4), and (5) are listed in *A/UX Programmer's Reference.*

For example,

```
cat(1)
```

refers to the command `cat`, which is described in Section 1 of *A/UX Command Reference*. References can also be called up on the screen. The `man` command or the `apropos` command displays pages from the reference manuals directly on the screen. For example, enter the command

```
man cat
```

In this example, the manual page for the `cat` command including its description, syntax, options, and other pertinent information appears on the screen. To exit, continue pressing the space bar until you see a command prompt, or press Q at any time to return immediately to your command prompt. The manuals often refer to information discussed in another guide in the suite. The format for this type of cross reference is "Chapter Title," *Name of Guide*. For a complete description of A/UX guides, see *Road Map to A/UX Documentation*. This guide contains descriptions of each A/UX guide, the part numbers, and the ordering information for all the guides in the A/UX documentation suite.

Introduction

to the A/UX Reference Manuals

1. How to use the reference manuals

A/UX Command Reference, *A/UX Programmer's Reference*, and *A/UX System Administrator's Reference* are reference manuals for all the programs, utilities, and standard file formats included with your A/UX® system.

The reference manuals constitute a compact encyclopedia of A/UX information. They are not intended to be tutorials or learning guides. If you are new to A/UX or are unfamiliar with a specific functional area (such as the shells or the text formatting programs), you should first read *A/UX Essentials* and the other A/UX user guides. After you have worked with A/UX, the reference manuals help you understand new features or refresh your memory about command features you already know.

2. Information contained in the reference manuals

A/UX reference manuals are divided into three volumes:

- The two-part *A/UX Command Reference* contains information for the general user. It describes commands you type at the A/UX prompt that list your files, compile programs, format text, change your shell, and so on. It also includes programs used in scripts and command language procedures. The commands in this manual generally reside in the directories `/bin`, `/usr/bin` and `/usr/ucb`.
- The two-part *A/UX Programmer's Reference* contains information for the programmer. It describes utilities for programming, such as system calls, file formats of subroutines, and miscellaneous programming facilities.
- *A/UX System Administrator's Reference* contains information for the system administrator. It describes commands you type at the A/UX prompt to control your machine, such as accounting

commands, backing up your system, and charting your system's activity. These commands generally reside in the directories `/etc`, `/usr/etc`, and `/usr/lib`.

These areas can overlap. For example, if you are the only person using your machine, then you are both the general user and the system administrator.

To help direct you to the correct manual, you may refer to *A/UX Reference Summary and Index*, which is a separate volume. This manual summarizes information contained in the other A/UX reference manuals. The three parts of this manual are a classification of commands by function, a listing of command synopses, and an index.

3. How the reference manuals are organized

All manual pages are grouped by section. The sections are grouped by general function and are numbered according to standard conventions as follows:

1	User commands
1M	System maintenance commands
2	System calls
3	Subroutines
4	File formats
5	Miscellaneous facilities
6	Games
7	Drivers and interfaces for devices
8	A/UX Startup shell commands

Manual pages are collated alphabetically by the primary name associated with each. For the individual sections, a table of contents is provided to show the sequence of manual pages. A notable exception to the alphabetical sequence of manual pages is the first entry at the start of each section. As a representative example, `intro.1` appears at the start of Section 1. These `intro.`*section-number* manual pages are brought to the front of each section because they introduce the other man pages in the same section, rather than describe a command or similar provision of A/UX.

Each of the reference manuals includes at least one complete section of man pages. For example, the *A/UX Command Reference* contains sections 1 and 6. However, since Section 1 (User Commands) is so large, this manual is divided into two volumes, the first containing Section 1 commands that begin with letters A through L, and the second containing Section 6 commands and Section 1 commands that begin with letters M through Z. The sections included in each volume are as follows.

A/UX Command Reference contains sections 1 and 6. Note that both of these sections describe commands and programs available to the general user.

- Section 1—User Commands
  The commands in Section 1 may also belong to a special category. Where applicable, these categories are indicated by the letter designation that follows the section number. For example, the N in `ypcat`(1N) indicates networking as described following.

1C	Communications commands, such as `cu` and `tip`.
1G	Graphics commands, such as `graph` and `tplot`.
1N	Networking commands, such as those which help support various networking subsystems, including the Network File System (NFS), Remote Process Control (RPC), and Internet subsystem.

- Section 6—User Commands
  This section contains all the games, such as `cribbage` and `worms`.

A/UX Programmer's Reference contains sections 2 through 5.

- Section 2—System Calls
  This section describes the services provided by the A/UX system kernel, including the C language interface. It includes two special categories. Where applicable, these categories are indicated by the letter designation that follows the section number. For example, the N in `connect`(2N) indicates networking as described following.

2N	Networking system calls
2P	POSIX system calls

- Section 3—Subroutines
  This section describes the available subroutines. The binary versions are in the system libraries in the `/lib` and `/usr/lib` directories. The section includes six special categories. Where applicable, these categories are indicated by the letter designation that follows the section number. For example, the N in `mount`(3N) indicates networking as described following.

3C	C and assembler library routines
3F	Fortran library routines
3M	Mathematical library routines
3N	Networking routines
2P	POSIX routines
3S	Standard I/O library routines
3X	Miscellaneous routines

- Section 4—File Formats
  This section describes the structure of some files, but does not include files that are used by only one command (such as the assembler's intermediate files). The C language `struct` declarations corresponding to these formats are in the `/usr/include` and `/usr/include/sys` directories. There is one special category in this section. Where applicable, these categories are indicated by the letter designation that follows the section number. For example, the N in

`protocols`(4N) indicates networking as described following.

4N Networking formats

- Section 5—Miscellaneous facilities
  This section contains various character sets, macro packages, and other miscellaneous formats. There are two special categories in this section. Where applicable, these categories are indicated by the letter designation that follows the section number. For example, the P in `tcp`(1P) indicates a protocol as described following. by the letter designation in parenthesis at the top of the page:

  5F Protocol families

  5P Protocol descriptions

A/UX System Administrator's Reference contains sections 1M, 7 and 8.

- Section 1M—System Maintenance Commands
  This section contains system maintenance programs such as `fsck` and `mkfs`.

- Section 7—Drivers and Interfaces for Devices
  This section discusses the drivers and interfaces through which devices are normally accessed. While access to one or more disk devices is fairly transparent when you are working with files, the provision of *device files* permits you more explicit modes with which to access particular disks or disk partitions, as well as other types of devices such as tape drives and modems. For example, a tape device may be accessed in automatic-rewind mode through one or more of the device file names in the `/dev/rmt` directory (see `tc`(7)). The **FILES** sections of these manual pages identify all the device files supplied with the system as well as those that are automatically generated by certain A/UX configuration utilities. The names of the man pages generally refer to device names or device driver names, rather than the names of the device files themselves.

- Section 8—A/UX Startup Shell Commands
  This section describes the commands that are available from within the A/UX Startup Shell, including detailed descriptions of

those that contribute to the boot process and those that help with the maintenance of file systems.

4. How a manual entry is organized

The name for a manual page entry normally appears twice, once in each upper corner of a page. Like dictionary guide words, these names appear at the top of every physical page. After each name is the section number and, if applicable, a category letter enclosed in parenthesis, such as (1) or (2N).

Some entries describe several routines or commands. For example, `chown` and `chgrp` share a page with the name `chown`(1) at the upper corners. If you turn to the page `chgrp`(1), you find a reference to `chown`(1). (These cross-reference pages are only included in *A/UX Command Reference* and *A/UX System Administrator's Reference*.)

All of the entries have a common format, and may include any of the following parts:

NAME
is the name or names and a brief description.

SYNOPSIS
describes the syntax for using the command or routine.

DESCRIPTION
discusses what the program does.

FLAG OPTIONS
discusses the flag options.

EXAMPLES
gives an example or examples of usage.

RETURN VALUE
describes the value returned by a function.

ERRORS
describes the possible error conditions.

FILES
lists the filenames that are used by the program.

SEE ALSO
provides pointers to related information.

DIAGNOSTICS
discusses the diagnostic messages that may be produced. Self-explanatory messages are not listed.

WARNINGS
points out potential pitfalls.

BUGS
gives known bugs and sometimes deficiencies. Occasionally, it describes the suggested fix.

5. Locating information in the reference manuals

The directory for the reference manuals, *A/UX Reference Summary and Index*, can help you locate information through its index and summaries. The tables of contents within each of the reference manuals can be used also.

5.1 Table of contents

Each reference manual contains an overall table of contents and individual section contents. The general table of contents lists the overall contents of each volume. The more detailed section contents lists the manual pages contained in each section and a brief description of their function. For the most part, entries appear in alphabetic order within each section.

5.2 Commands by function

This summary classifies the A/UX user and administration commands by the general, or most important function they perform. The complete descriptions of these commands are found in *A/UX Command Reference* and *A/UX System Administrator's Reference*. Each is mentioned just once in this listing.

The summary gives you a broader view of the commands that are available and the context in which they are most often used.

5.3 Command synopses

This section is a compact collection of syntax descriptions for all the commands in *A/UX Command Reference* and *A/UX System Administrator's Reference*. It may help you find the syntax of commands more quickly when the syntax is all you need.

5.4 Index

The index lists key terms associated with A/UX subroutines and commands. These key terms allow you to locate an entry when you don't know the command or subroutine name.

The key terms were constructed by examining the meaning and usage of the A/UX manual pages. It is designed to be more discriminating and easier to use than the traditional permuted index, which lists nearly all words found in the manual page NAME sections.

Most manual pages are indexed under more than one entry; for example, `lorder`(1) is included under "archive files," "sorting," and "cross-references." This way you are more likely to find the reference you are looking for on the first try.

5.5 Online documentation

Besides the paper documentation in the reference manuals, A/UX provides several ways to search and read the contents of each reference from your A/UX system.

To see a manual page displayed on your screen, enter the `man`(1) command followed by the name of the entry you want to see. For example,

```
man passwd
```

To see the description phrase from the NAME section of any manual page, enter the `whatis` command followed by the name of the entry you want to see. For example,

```
whatis apropos
```

To see a list of all manual pages whose descriptions contain a given keyword or string, enter the `apropos` command followed by the word or string. For example,

```
apropos remove
```

These online documentation commands are described more fully in the manual pages `man`(1), `whatis`(1), and `apropos`(1) in *A/UX Command Reference*.

Table of Contents

Section 3: Subroutines (M-Z)

mallinfo(3X) .... see malloc(3X)
malloc(3C) .... main memory allocator
malloc(3X) .... fast main memory allocator
mallopt(3X) .... see malloc(3X)
matherr(3M) .... error-handling function
max(3F) .... Fortran maximum-value functions
max0(3F) .... see max(3F)
max1(3F) .... see max(3F)
mclock(3F) .... return Fortran time accounting
memccpy(3C) .... see memory(3C)
memchr(3C) .... see memory(3C)
memcmp(3C) .... see memory(3C)
memcpy(3C) .... see memory(3C)
memory(3C) .... memory operations
memset(3C) .... see memory(3C)
min(3F) .... Fortran minimum-value functions
min0(3F) .... see min(3F)
min1(3F) .... see min(3F)
mkfifo(3P) .... make a FIFO special file
mktemp(3C) .... make a unique filename
mktime(3) .... see ctime(3)
mod(3F) .... Fortran remaindering intrinsic functions
modf(3C) .... see frexp(3C)
monitor(3C) .... prepare execution profile
mount(3) .... mount a file system
mount(3N) .... keep track of remotely mounted file systems
mrand48(3C) .... see drand48(3C)
nbp(3N) .... AppleTalk Name Binding Protocol (NBP) interface.
nbp_confirm(3N) .... see nbp(3N)
nbp_lookup(3N) .... see nbp(3N)
nbp_make_entity(3N) .... see nbp(3N)
nbp_parse_entity(3N) .... see nbp(3N)
nbp_register(3N) .... see nbp(3N)
nbp_remove(3N) .... see nbp(3N)
nextkey(3X) .... see dbm(3X)
nint(3F) .... see round(3F)
nlist(3C) .... get entries from name list

not(3F) .... see bool(3F)
nrand48(3C) .... see drand48(3C)
ntohl(3N) .... see byteorder(3N)
ntohs(3N) .... see byteorder(3N)
numbptabent(3) .... see getptabent(3)
opendir(3) .... see directory(3)
opendir(3P) .... see directory(3P)
or(3F) .... see bool(3F)
pap(3N) .... AppleTalk Printer Access Protocol (PAP) interface
paps_close(3N) .... see pap(3N)
paps_get_next_job(3N) .... see pap(3N)
paps_open(3N) .... see pap(3N)
paps_status(3N) .... see pap(3N)
pap_close(3N) .... see pap(3N)
pap_open(3N) .... see pap(3N)
pap_read(3N) .... see pap(3N)
pap_read_ignore(3N) .... see pap(3N)
pap_status(3N) .... see pap(3N)
pap_write(3N) .... see pap(3N)
pathconf(3P) .... get configurable pathname variables
pclose(3S) .... see popen(3S)
perror(3C) .... system error messages
plot(3X) .... graphics interface subroutines
popen(3S) .... initiate pipe to/from a process
pow(3M) .... see exp(3M)
printf(3S) .... format and output string and numeric data
putc(3S) .... put character or word on a stream
putchar(3S) .... see putc(3S)
putenv(3C) .... change or add value to environment
putpwent(3C) .... write password file entry
puts(3S) .... put a string on a stream
pututline(3C) .... see getut(3C)
putw(3S) .... see putc(3S)
qsort(3C) .... quicker sort
rand(3C) .... simple random-number generator
rand(3F) .... Fortran uniform random-number generator
rcmd(3N) .... routines for returning a stream to a remote command
readdir(3) .... see directory(3)
readdir(3P) .... see directory(3P)
real(3F) .... see ftype(3F)
realloc(3C) .... see malloc(3C)
realloc(3X) .... see malloc(3X)
regcmp(3X) .... compile and execute a regular expression

regex(3X) .... see regcmp(3X)
remque(3N) .... see insque(3N)
resolver(3N) .... resolver routines
res_init(3N) .... see resolver(3N)
res_mkquery(3N) .... see resolver(3N)
res_send(3N) .... see resolver(3N)
rewind(3S) .... see fseek(3S)
rewinddir(3) .... see directory(3)
rewinddir(3P) .... see directory(3P)
rexec(3N) .... return stream to a remote command
rnusers(3N) .... return information about users on remote machines
round(3F) .... Fortran nearest integer functions
rpc(3N) .... library routines for remote procedure calls
rresvport(3N) .... see rcmd(3N)
rshift(3F) .... see bool(3F)
rtmp(3N) .... identify AppleTalk node and bridge addresses
rtmp_netinfo(3N) .... see rtmp(3N)
ruserok(3N) .... see rcmd(3N)
rusers(3N) .... see rnusers(3N)
rwall(3N) .... write to specified remote machines
scandir(3) .... scan a directory
scanf(3S) .... convert formatted input
seed48(3C) .... see drand48(3C)
seekdir(3) .... see directory(3)
seekdir(3P) .... see directory(3P)
set42sig(3) .... set 4.2 BSD signal interface
setbuf(3S) .... assign buffering to a stream
setgid(3) .... see setuid(3)
setgrent(3C) .... see getgrent(3C)
setjmp(3C) .... non-local goto
setmntent(3) .... see getmntent(3)
setnetent(3N) .... see getnetent(3N)
setnetgrent(3N) .... see getnetgrent(3N)
setposix(3P) .... set POSIX compatibility flags
setprotoent(3N) .... see getprotoent(3N)
setptabent(3) .... see getptabent(3)
setpwent(3C) .... see getpwent(3C)
setrpcent(3N) .... see getrpcent(3N)
setservent(3N) .... see getservent(3N)
setuid(3) .... set user and group IDs
setutent(3C) .... see getut(3C)
setvbuf(3S) .... see setbuf(3S)
sgetl(3X) .... see sputl(3X)

sigaction(3P) .... examine or change signal action
sigaddset(3P) .... see sigsetops(3P)
sigdelset(3P) .... see sigsetops(3P)
sigfillset(3P) .... see sigsetops(3P)
siginitset(3P) .... see sigsetops(3P)
sigismember(3P) .... see sigsetops(3P)
siglongjmp(3P) .... see sigsetjmp(3P)
sign(3F) .... Fortran transfer-of-sign intrinsic function
signal(3) .... specify what to do upon receipt of a signal
signal(3F) .... specify Fortran action on receipt of a system signal
sigprocmask(3P) .... examine and change blocked signals
sigsetjmp(3P) .... non-local jumps
sigsetops(3P) .... manipulate signal sets
sigsuspend(3P) .... wait for a signal
sin(3F) .... Fortran sine intrinsic function
sin(3M) .... see trig(3M)
sinh(3F) .... Fortran hyperbolic sine intrinsic function
sinh(3M) .... hyperbolic functions
sleep(3C) .... suspend execution for interval
slots(3X) .... ROM library functions
sngl(3F) .... see ftype(3F)
spray(3N) .... scatter data in order to check the network
sprintf(3S) .... see printf(3S)
sputl(3X) .... access long integer data in a machine independent fashion
sqrt(3F) .... Fortran square root intrinsic function
sqrt(3M) .... see exp(3M)
srand(3C) .... see rand(3C)
srand(3F) .... see rand(3F)
srand48(3C) .... see drand48(3C)
sscanf(3S) .... see scanf(3S)
ssignal(3C) .... software signals
store(3X) .... see dbm(3X)
strcat(3C) .... see string(3C)
strchr(3C) .... see string(3C)
strcmp(3C) .... see string(3C)
strcpy(3C) .... see string(3C)
strcspn(3C) .... see string(3C)
string(3C) .... string operations
strlen(3C) .... see string(3C)
strncat(3C) .... see string(3C)
strncmp(3C) .... see string(3C)
strncpy(3C) .... see string(3C)
strpbrk(3C) .... see string(3C)

strrchr(3C) .... see string(3C)
strspn(3C) .... see string(3C)
strtod(3C) .... convert string to double-precision number
strtok(3C) .... see string(3C)
strtol(3C) .... convert string to integer
swab(3C) .... swap bytes
sysconf(3P) .... get configurable system variables
system(3F) .... issue a shell command from Fortran
system(3S) .... issue a shell command
sys_errlist(3C) .... see perror(3C)
sys_nerr(3C) .... see perror(3C)
tan(3F) .... Fortran tangent intrinsic function
tan(3M) .... see trig(3M)
tanh(3F) .... Fortran hyperbolic tangent intrinsic function
tanh(3M) .... see sinh(3M)
tcdrain(3P) .... line control functions
tcflow(3P) .... see tcdrain(3P)
tcflush(3P) .... see tcdrain(3P)
tcgetattr(3P) .... get and set the terminal state
tcgetpgrp(3P) .... get distinguished process group ID
tcsendbreak(3P) .... see tcdrain(3P)
tcsetattr(3P) .... see tcgetattr(3P)
tcsetpgrp(3P) .... set distinguished process group ID
tdelete(3C) .... see tsearch(3C)
telldir(3) .... see directory(3)
telldir(3P) .... see directory(3P)
tempnam(3S) .... see tmpnam(3S)
termcap(3X) .... terminal independent operation routines
tfind(3C) .... see tsearch(3C)
tgetent(3X) .... see termcap(3X)
tgetflag(3X) .... see termcap(3X)
tgetnum(3X) .... see termcap(3X)
tgetstr(3X) .... see termcap(3X)
tgoto(3X) .... see termcap(3X)
tmpfile(3S) .... create a temporary file
tmpnam(3S) .... create a name for a temporary file
toascii(3C) .... see conv(3C)
tolower(3C) .... see conv(3C)
toupper(3C) .... see conv(3C)
tputs(3X) .... see termcap(3X)
trig(3M) .... trigonometric functions
tsearch(3C) .... manage binary search trees
ttyname(3C) .... find name of a terminal

ttyslot(3C) .... find the slot in the utmp file of the current user
twalk(3C) .... see tsearch(3C)
tzset(3) .... see ctime(3)
tzsetwall(3) .... see ctime(3)
umount(3) .... unmount a file system
ungetc(3S) .... push character back into input stream
utmpname(3C) .... see getut(3C)
varargs(3X) .... handle variable argument list
vfprintf(3S) .... see vprintf(3S)
vprintf(3S) ..... format and output data from a variable-length argument list
vsprintf(3S) .... see vprintf(3S)
xdr(3N) .... library routines for external data representation
xor(3F) .... see bool(3F)
y0(3M) .... see bessel(3M)
y1(3M) .... see bessel(3M)
yn(3M) .... see bessel(3M)
ypclnt(3N) .... yellow pages client interface
yperr_string(3N) .... see ypclnt(3N)
yppasswd(3N) .... update user password in yellow pages
ypprot_err(3N) .... see ypclnt(3N)
yp_all(3N) .... see ypclnt(3N)
yp_bind(3N) .... see ypclnt(3N)
yp_first(3N) .... see ypclnt(3N)
yp_get_default_domain(3N) .... see ypclnt(3N)
yp_master(3N) .... see ypclnt(3N)
yp_match(3N) .... see ypclnt(3N)
yp_next(3N) .... see ypclnt(3N)
yp_order(3N) .... see ypclnt(3N)
yp_unbind(3N) .... see ypclnt(3N)
zabs(3F) .... see abs(3F)
zip(3N) .... AppleTalk Zone Information Protocol (ZIP) interface
zip_getlocalzones(3N) .... see zip(3N)
zip_getmyzone(3N) .... see zip(3N)
zip_getzonelist(3N) .... see zip(3N)
_tolower(3C) .... see conv(3C)
_toupper(3C) .... see conv(3C)

NAME

malloc, free, realloc, calloc, cfree — main memory allocator

SYNOPSIS

```
char *malloc(size)
unsigned size;

void free(ptr)
char *ptr;

char *realloc(ptr, size)
char *ptr;
unsigned size;

char *calloc(nelem, elsize)
unsigned nelem, elsize;

void cfree(ptr, nelem, elsize)
char *ptr;
unsigned nelem, elsize;
```

DESCRIPTION

malloc and free provide a simple general-purpose memory allocation package. malloc returns a pointer to a block of at least *size* bytes suitably aligned for any use.

The argument to free is a pointer to a block previously allocated by malloc; after free is performed, this space is made available for further allocation, but its contents are left undisturbed.

Undefined results occur if the space assigned by malloc is overrun or if some random number is handed to free.

malloc allocates the first contiguous reach of free space of sufficient size found in a circular search from the last block allocated or freed; it coalesces adjacent free blocks as it searches. It calls sbrk (see brk(2)) to get more memory from the system when there is no suitable space already free.

realloc changes the size of the block pointed to by *ptr* to *size* bytes and returns a pointer to the (possibly moved) block. The contents are unchanged up to the lesser of the new and old sizes. If no free block of *size* bytes is available in the storage arena, realloc asks malloc to enlarge the arena by *size* bytes and then moves the data to the new space.

realloc also works if *ptr* points to a block freed since the last call of malloc, realloc, or calloc; thus sequences of free, malloc, and realloc can exploit the search strategy of malloc to do storage compaction.

calloc allocates space for an array of *nelem* elements of size *elsize*. The space is initialized to zeros.

The arguments to cfree are the pointer to a block previously allocated by calloc plus the parameters to calloc.

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.

RETURN VALUE

malloc, realloc, and calloc return a NULL pointer if there is no available memory or if the arena is deteected to have been corrupted by storing outside the bounds of a block. When this happens the block pointed to by *ptr* may be destroyed.

NOTES

Search time increases when many objects have been allocated; that is, if a program allocates space but never frees it, each successive allocation takes longer.

SEE ALSO

brk(2), malloc(3X).

NAME

malloc, free, realloc, calloc, mallopt, mallinfo — fast main memory allocator

SYNOPSIS

```
#include <malloc.h>

char *malloc(size)
unsigned size;

void free(ptr)
char *ptr;

char *realloc(ptr, size)
char *ptr;
unsigned size;

char *calloc(nelem, elsize)
unsigned nelem, elsize;

int mallopt(cmd, value)
int cmd, value;

struct mallinfo mallinfo(max)
int max;
```

DESCRIPTION

`malloc` and `free` provide a simple general-purpose memory allocation package, which runs considerably faster than the `malloc`(3C) package. It is found in the library "`malloc`", and is loaded if the option "`-lmalloc`" is used with `cc`(1) or `ld`(1).

`malloc` returns a pointer to a block of at least *size* bytes suitably aligned for any use.

The argument to `free` is a pointer to a block previously allocated by `malloc`; after `free` is performed this space is made available for further allocation, and its contents have been destroyed (but see `mallopt` below for a way to change this behavior).

Undefined results will occur if the space assigned by `malloc` is overrun or if some random number is handed to `free`.

`realloc` changes the size of the block pointed to by *ptr* to *size* bytes and returns a pointer to the (possibly moved) block. The contents will be unchanged up to the lesser of the new and old sizes.

`calloc` allocates space for an array of *nelem* elements of size *elsize*. The space is initialized to zeros.

`mallopt` provides for control over the allocation algorithm. The available values for *cmd* are:

`M_MXFAST` Set `maxfast` to *value*. The algorithm allocates all blocks below the size of `maxfast` in large groups and then doles them out very quickly. The default value for `maxfast` is 0.

`M_NLBLKS` Set `numlblks` to *value*. The above mentioned "large groups" each contain `numlblks` blocks. `numlblks` must be greater than 0. The default value for `numlblks` is 100.

`M_GRAIN` Set `grain` to *value*. The sizes of all blocks smaller than `maxfast` are considered to be rounded up to the nearest multiple of `grain`. `grain` must be greater than 0. The default value of `grain` is the smallest number of bytes which will allow alignment of any data type. Value will be rounded up to a multiple of the default when `grain` is set.

`M_KEEP` Preserve data in a freed block until the next `malloc`, `realloc`, or `calloc`. This option is provided only for compatibility with the old version of `malloc` and is not recommended.

These values are defined in the `<malloc.h>` header file.

`mallopt` may be called repeatedly, but may not be called after the first small block is allocated.

`mallinfo` provides instrumentation describing space usage. It returns the structure:

```
struct mallinfo  {
    int arena;     /* total space in arena */
    int ordblks;   /* number of ordinary blocks */
    int smblks;    /* number of small blocks */
    int hblkhd;    /* space in holding block headers */
    int hblks;     /* number of holding blocks */
    int usmblks;   /* space in small blocks in use */
    int fsmblks;   /* space in free small blocks */
    int uordblks;  /* space in ordinary blocks in use */
    int fordblks;  /* space in free ordinary blocks */
```

```
    int keepcost;  /* space penalty if keep option */
                   /* is used */
}
```

This structure is defined in the <malloc.h> header file.

Each of the allocation routines returns a pointer to space suitably aligned (after possible pointer coercion) for storage of any type of object.

RETURN VALUE

`malloc`, `realloc` and `calloc` return a NULL pointer if there is not enough available memory. When `realloc` returns NULL, the block pointed to by *ptr* is left intact. If `mallopt` is called after any allocation or if *cmd* or *value* are invalid, non-zero is returned. Otherwise, it returns zero.

SEE ALSO

`brk`(2), `malloc`(3C).

WARNINGS

This package usually uses more data space than `malloc`(3C).
The code size is also bigger than `malloc`(3C).
Note that unlike `malloc`(3C), this package does not preserve the contents of a block when it is freed, unless the `M_KEEP` option of `mallopt` is used.
Undocumented features of `malloc`(3C) have not been duplicated.

NAME

matherr — error-handling function

SYNOPSIS

```
#include <math.h>

int matherr(x)
struct exception *x;
```

DESCRIPTION

matherr is invoked by functions in the Math Library when errors are detected. Users may define their own procedures for handling errors, by including a function named matherr in their programs. matherr must be of the form described above. When an error occurs, a pointer to the exception structure *x* will be passed to the user-supplied matherr function. This structure, which is defined in the <math.h> header file, is as follows:

```
struct exception {
        int type;
        char *name;
        double arg1;
        double arg2;
        double retval;
};
```

The element *type* is an integer describing the type of error that has occurred, from the following list of constants (defined in the header file):

DOMAIN	argument domain error
SING	argument singularity
OVERFLOW	overflow range error
UNDERFLOW	underflow range error
TLOSS	total loss of significance
PLOSS	partial loss of significance

The element *name* points to a string containing the name of the function that incurred the error. The variables *arg1* and *arg2* are the arguments with which the function was invoked. *retval* is set to the default value that will be returned by the function unless the user's matherr sets it to a different value.

If the user's matherr function returns nonzero, no error message will be printed, and errno will not be set.

If matherr is not supplied by the user, the default error-handling procedures, described with the math functions involved, will be invoked upon error. These procedures are also summarized in the table below. In every case, errno is set to EDOM or ERANGE and the program continues.

EXAMPLES

```
#include <math.h>

int
matherr(x)
register struct exception *x;
{
   switch (x->type) {
   case DOMAIN:
        /* change sqrt to return sqrt(-arg1), not 0 */
        if (!strcmp(x->name, "sqrt")) {
            x->retval = sqrt(-x->arg1);
            return (0); /* print message and set errno */
        }
   case SING:
        /* all other domain or sing errors,
           print message and abort */
        fprintf(stderr, "domain error in %s\n", x->name);
        abort( );
   case PLOSS:
        /* print detailed error message */
        fprintf(stderr, "loss of significance in %s(%g) = %g\n",
            x->name, x->arg1, x->retval);
        return (1); /* take no other action */
   }
        return (0); /* all other errors,
                       execute default procedure */
}
```

DEFAULT ERROR HANDLING PROCEDURES

	Types of Errors					
type	DOMAIN	SING	OVERFLOW	UNDERFLOW	TLOSS	PLOSS
errno	EDOM	EDOM	ERANGE	ERANGE	ERANGE	ERANGE
BESSEL:	–	–	–	–	M, 0	*
y0, y1, yn (arg ≤ 0)	M, –H	–	–	–	–	–
EXP:	–	–	H	0	–	–
LOG, LOG10:						
(arg < 0)	M, –H	–	–	–	–	–
(arg = 0)	–	M, –H	–	–	–	–
POW:	–	–	±H	0	–	–
neg ** nonint	M, 0	–	–	–	–	–
0 ** nonpos						
SQRT:	M, 0	–	–	–	–	–
GAMMA:	–	M, H	H	–	–	–
HYPOT:	–	–	H	–	–	–
SINH:	–	–	±H	–	–	–
COSH:	–	–	H	–	–	–
SIN, COS, TAN:	–	–	–	–	M, 0	*
ASIN, ACOS, ATAN2:	M, 0	–	–	–	–	–

ABBREVIATIONS

*	As much as possible of the value is returned.
M	Message is printed (EDOM error).
H	HUGE is returned.
–H	–HUGE is returned.
±H	HUGE or –HUGE is returned.
0	0 is returned.

NAME

max, max0, amax0, max1, amax1, dmax1 — Fortran maximum-value functions

SYNOPSIS

```
integer i, j, k, l
real a, b, c, d
double precision dp1, dp2, dp3

l=max(i, j, k)
c=max(a, b)
d=max(a, b, c)
k=max0(i, j)
a=amax0(i, j, k)
i=max1(a, b)
d=amax1(a, b, c)
dp3=dmax1(dp1, dp2)
```

DESCRIPTION

The maximum-value functions return the largest of their arguments; there may be any number of arguments. max is the generic form which can be used for all data types and takes its return type from that of its arguments. All arguments must be of the same type. max0 returns the integer form of the maximum value of its integer arguments; amax0, the real form of its integer arguments; max1, the integer form of its real arguments; amax1, the real form of its real arguments; and dmax1, the double-precision form of its double-precision arguments.

SEE ALSO

min(3F).

NAME

`mclock` — return Fortran time accounting

SYNOPSIS

`integer` *i*

i=`mclock()`

DESCRIPTION

`mclock` returns time accounting information about the current process and its child processes. The value returned is the sum of the current process's user time and the user and system times of all child processes.

SEE ALSO

`times`(2), `clock`(3C), `system`(3F).

NAME

memccpy, memchr, memcmp, memcpy, memset — memory operations

SYNOPSIS

```
#include <memory.h>

char *memccpy(s1, s2, c, n)
char *s1, *s2;
int c, n;

char *memchr(s, c, n)
char *s;
int c, n;

int memcmp(s1, s2, n)
char *s1, *s2;
int n;

char *memcpy(s1, s2, n)
char *s1, *s2;
int n;

char *memset(s, c, n)
char *s;
int c, n;
```

DESCRIPTION

These functions operate efficiently on memory areas (arrays of characters bounded by a count, not terminated by a null character). They do not check for the overflow of any receiving memory area.

memccpy copies characters from memory area *s2* into *s1*, stopping after the first occurrence of character *c* has been copied or after *n* characters have been copied, whichever comes first. It returns either a pointer to the character after the copy of *c* in *s1* or a NULL pointer if *c* was not found in the first *n* characters of *s2*.

memchr returns either a pointer to the first occurrence of character *c* in the first *n* characters of memory area *s* or a NULL pointer if *c* does not occur.

memcmp compares its arguments, looking at the first *n* characters only. It returns an integer less than, equal to, or greater than 0, depending on whether *s1* is lexicographically less than, equal to, or greater than *s2*.

memcpy copies *n* characters from memory area *s2* to *s1*. It returns *s1*.

memset sets the first *n* characters in memory area *s* to the value of character *c*. It returns *s*.

NOTES

For user convenience, all these functions are declared in the optional <memory.h> header file.

BUGS

memcmp uses native character comparison.

Because character movement is performed differently in different implementations, overlapping moves may yield unexpected results.

NAME

`min`, `min0`, `amin0`, `min1`, `amin1`, `dmin1` — Fortran minimum-value functions

SYNOPSIS

integer *i*, *j*, *k*, *l*
real *a*, *b*, *c*, *d*
double precision *dp1*, *dp2*, *dp3*

l=min(*i*, *j*, *k*)
c=min(*a*, *b*)
d=min(*a*, *b*, *c*)
k=min0(*i*, *j*)
a=amin0(*i*, *j*, *k*)
i=min1(*a*, *b*)
d=amin1(*a*, *b*, *c*)
dp3=dmin1(*dp1*, *dp2*)

DESCRIPTION

The minimum-value functions return the minimum of their arguments. There may be any number of arguments. `min` is the generic form which can be used for all data types. It takes its return type from that of its arguments, which must all be of the same type. `min0` returns the integer form of the minimum value of its integer arguments; `amin0`, the real form of its integer arguments; `min1`, the integer form of its real arguments; `amin1`, the real form of its real arguments; and `dmin1`, the double-precision form of its double-precision arguments.

SEE ALSO

max(3F).

NAME

mkfifo — make a FIFO special file

SYNOPSIS

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

int mkfifo(path, mode)
char *path;
mode_t mode;
```

DESCRIPTION

mkfifo creates a new FIFO special file named by the pathname pointed to by *path*. The mode of the new FIFO is initialized from *mode*. The file permission bits of *mode* are modified by the file creation mask of the process. If bits in *mode* other than file permissions are set, the permissions on the FIFO will be undefined.

For the POSIX environment, the following constants for *mode* are defined in <sys/stat.h> :

S_IRUSR	read permission, owner
S_IWUSER	writer permission, owner
S_IXUSR	execute/search permission, owner
S_IRGRP	read permission, group
S_IWGRP	writer permission, group
S_IXGRP	execute/search permission, group
S_IROTH	read permission, others
S_IWOTH	writer permission, others
S_IXOTH	execute/search permission, others

The owner ID of the FIFO is set to the effective user ID of the process. The group ID of the FIFO is set to the effective group ID of the process.

On successful completion, mkfifo marks for update the st_atime, st_ctime, and st_mtime fields for the file. The st_ctime and st_mtime fields of the directory that contains the new entry are also marked for update.

RETURN VALUE

On successful completion, mkfifo returns a value of 0. Otherwise, a value of −1 is returned, no FIFO is created, and errno is set to indicate the error.

ERRORS

mkfifo will fail and the new FIFO will not be created if one or more of the following are true:

[ENAMETOOLONG]	A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX.
[ELOOP]	Too many symbolic links were encountered in translating a pathname.
[ENOTDIR]	A component of the path prefix is not a directory.
[ENOENT]	A component of the path prefix does not exist.
[EROFS]	The directory in which the FIFO is to be created is located on a read-only file system.
[EEXIST]	The named FIFO exists.
[EFAULT]	*path* points outside the allocated address space of the process.

SEE ALSO

mknod(2), umask(2).

NAME

mktemp — make a unique filename

SYNOPSIS

`char *mktemp(`*template*`)`
`char *`*template*`;`

DESCRIPTION

The function mktemp alters the contents of the string referenced by **template* so that it becomes a unique filename. The string at **template* should be initialized to a filenamed with six trailing X characters; mktemp replaces the Xs with a letter and the current process ID. The letter is selected so that the resulting name is not a duplicate an existing file.

RETURN VALUE

mktemp returns the address of the unique (altered) filename. If a unique name cannot be created, template will point to a null (empty) string.

SEE ALSO

getpid(2), tmpfile(3S), tmpnam(3S).

BUGS

It is possible to run out of letters.

NAME

mod, amod, dmod — Fortran remaindering intrinsic functions

SYNOPSIS

```
integer i, j, k
real r1, r2, r3
double precision dp1, dp2, dp3

k=mod(i, j)

r3=amod(r1, r2)
r3=mod(r1, r2)

dp3=dmod(dp1, dp2)
dp3=mod(dp1, dp2)
```

DESCRIPTION

mod returns the integer remainder of its first argument divided by its second argument. amod and dmod return, respectively, the real and double-precision whole number remainder of the integer division of their two arguments. The generic version mod returns the data type of its arguments.

NAME

monitor — prepare execution profile

SYNOPSIS

```
#include <mon.h>

void monitor(lowpc, highpc, buffer, bufsize, nfunc)
int (*lowpc) (), (*highpc) ();
WORD *buffer;
int bufsize, nfunc;
```

DESCRIPTION

An executable program created by `cc -p` automatically includes calls for `monitor` with default parameters; `monitor` needn't be called explicitly except to gain fine control over profiling.

`monitor` is an interface to `profil`(2). *lowpc* and *highpc* are the addresses of two functions; *buffer* is the address of a (user supplied) array of *bufsize* elements of type `WORD` (defined in the `<mon.h>` header file). `monitor` arranges to record a histogram in the buffer. This histogram shows periodically sampled values of the program counter and counts of calls of certain functions. The lowest address sampled is that of *lowpc*; the highest address is just below *highpc*. *lowpc* may not equal 0 for this use of `monitor`. *nfunc* is the maximum number of call counts that can be kept; only calls of functions compiled with the profiling option `-p` of `cc`(1) are recorded. (The C Library and Math Library supplied when `cc -p` is used also have call counts recorded.) For the results to be significant, especially where there are small, heavily used routines, it is suggested that the buffer be no more than a few times smaller than the range of locations sampled.

To profile the entire program, it is sufficient to use:

```
extern etext;
monitor((int (*)())2, etext, buf, bufsize, nfunc);
```

`etext` lies just above all the program text; see `end`(3C).

To stop execution monitoring and write the results on the file `mon.out`, use

```
monitor ((int (*)())0, 0, 0, 0, 0);
```

`prof`(1) can then be used to examine the results.

FILES

mon.out
/lib/libp/libc.a
/lib/libp/libm.a

SEE ALSO

cc(1), prof(1), profil(2), end(3C).

NAME

`mount` — mount a file system

SYNOPSIS

`int mount(`*spec*`,` *dir*`,` *rwflag*`)`
`char *`*spec*`, *`*dir*`;`
`int` *rwflag*`;`

DESCRIPTION

`mount` requests that a removable file system contained on the block special file identified by *spec* be mounted on the directory identified by *dir*. *spec* and *dir* are pointers to path names.

Upon successful completion, references to the file *dir* will refer to the root directory on the mounted file system.

The low-order bit of *rwflag* is used to control write permission on the mounted file system; if `1`, writing is forbidden, otherwise writing is permitted according to individual file accessibility. Physically write-protected and magnetic tape file systems must be mounted read-only or errors will occur when access times are updated, whether or not any explicit write is attempted.

`mount` may be invoked only by the superuser.

ERRORS

`mount` will fail if one or more of the following are true:

[EPERM]	The effective user ID is not superuser.
[ENOENT]	Any of the named files does not exist.
[ENOTDIR]	A component of a path prefix is not a directory.
[ENOTBLK]	*spec* is not a block special device.
[ENXIO]	The device associated with *spec* does not exist.
[ENOTDIR]	*dir* is not a directory.
[EFAULT]	*spec* or *dir* points outside the allocated address space of the process.
[EBUSY]	*dir* is currently mounted on, is someone's current working directory, or is otherwise busy.
[EPERM]	A pathname contains a character with the high-order bit set.

[ENAMETOOLONG]	A component of a pathname exceeded NAME_MAX characters, or an entire pathname exceeded PATH_MAX.
[ELOOP]	Too many symbolic links were encountered in translating a pathname.
[EBUSY]	The device associated with *spec* is currently mounted.
[EBUSY]	There are no more mount table entries.

RETURN VALUE

Upon successful completion a value of 0 is returned. Otherwise, a value of −1 is returned and errno is set to indicate the error.

SEE ALSO

fsmount(2), unmount(2), umount(3), fstab(4).

NAME

mount — keep track of remotely mounted file systems

SYNOPSIS

```
#include <rpcsvc/mount.h>
```

DESCRIPTION

RPC INFO

Program number: MOUNTPROG

xdr routines:

```
xdr_exportbody(xdrs, ex)
    XDR *xdrs;
    struct exports *ex;
xdr_exports(xdrs, ex);
    XDR *xdrs;
    struct exports **ex;
xdr_fhandle(xdrs, fh);
    XDR *xdrs;
    fhandle_t *fp;
xdr_fhstatus(xdrs, fhs);
    XDR *xdrs;
    struct fhstatus *fhs;
xdr_groups(xdrs, gr);
    XDR *xdrs;
    struct groups *gr;
xdr_mountbody(xdrs, ml)
    XDR *xdrs;
    struct mountlist *ml;
xdr_mountlist(xdrs, ml);
    XDR *xdrs;
    struct mountlist **ml;
xdr_path(xdrs, path);
    XDR *xdrs;
    char **path;
```

Procs:

MOUNTPROC_MNT

Argument of xdr_path; returns fhstatus. Requires UNIX authentication.

MOUNTPROC_DUMP

No arguments; returns structure mountlist.

MOUNTPROC_UMNT

Argument of xdr_path; no results. Requires UNIX authentication.

MOUNTPROC_UMNTALL

No arguments; no results. Requires UNIX authentication. Unmounts all remote mounts of sender.

MOUNTPROC_EXPORT
MOUNTPROC_EXPORTALL

No arguments; returns structure exports.

Versions: MOUNTVERS_ORIG

Structures:

```
struct mountlist { /* what is mounted */
   char *ml_name;
   char *ml_path;
   struct mountlist *ml_nxt;
};
struct fhstatus {
   int fhs_status;
   fhandle_t fhs_fh;
};
/*
 * List of exported directories
 * An export entry with ex_groups NULL
 * indicates an entry which is exported
 * to the world.
 */
struct exports {
   dev_t ex_dev;  /* dev of directory */
   char *ex_name; /* name of directory */
   struct groups *ex_groups;
   /* groups allowed to mount this entry */
   struct exports *ex_next;
};
struct groups {
   char *g_name;
   struct groups *g_next;
};
```

SEE ALSO

mount(1M), mountd(1M), showmount(1M). *NFS Protocol Spec*, Section 3, in *A/UX Network Applications Programming*.

NAME

nbp_parse_entity, nbp_make_entity, nbp_confirm, nbp_lookup, nbp_register, nbp_remove — AppleTalk Name Binding Protocol (NBP) interface.

SYNOPSIS

```
#include <at/appletalk.h>
#include <at/nbp.h>
cc [flags] files -lat [libraries]

int nbp_parse_entity(entity, str);
at_entity_t *entity;
char *str;

int nbp_make_entity(entity, object, type, zone);
at_entity_t *entity;
char *object, *type, *zone;

int nbp_confirm(entity, dest, retry);
at_entity_t *entity;
at_inet_t *dest;
at_retry_t *retry;

int nbp_lookup(entity, buf, max, retry);
at_entity_t *entity;
at_nbptuple_t *buf;
int max;
at_retry_t *retry;

int nbp_register(entity, fd, retry);
at_entity_t *entity;
int fd;
at_retry_t *retry;

int nbp_remove(entity, fd );
at_entity_t *entity;
int fd;
```

DESCRIPTION

The NBP interface provides applications with access to the NBP operations. The routines use these structures (defined in <at/appletalk.h>):

```
typedef struct at_inet {
        at_net      net;
        at_node     node;
```

```
        at_socket  socket;
} at_inet_t;

typedef struct at_retry {
        short    interval;
        short    retries;
        u_char   backoff;
} at_retry_t;
```

The AppleTalk NBP operations also use these structures (defined in <at/nbp.h>):

```
typedef struct at_nvestr {
        char      len;
        char      str[NBP_NVE_STR_SIZE];
} at_nvestr_t;

typedef struct at_entity {
        at_nvestr_t   object;
        at_nvestr_t   type;
        at_nvestr_t   zone;
} at_entity_t;

typedef struct at_nbptuple {
        at_inet_t     enu_addr;
        u_char        enu_enum;
        at_entity_t   enu_entity;
} at_nbptuple_t;
```

The `at_inet_t` structure specifies the AppleTalk internet address of a DDP socket endpoint.

The `at_retry_t` structure specifies the retry interval and maximum count for a transaction. The members of this structure are

interval The interval in seconds before NBP retries a request.

retries The maximum number of retries for this NBP request.

backoff Not used by NBP.

The `at_nvestr_t` structure specifies an NBP entity string. The members of this structure are:

len The length of the string in bytes.

str The character data for this string.

The `at_entity_t` structure describes an entity name, which consists of three NBP entity strings: *object, type,* and *zone.*

All NBP routines work with the `at_entity_t` structure. Two utility routines, `nbp_parse_entity`, and `nbp_make_entity`, are provided to aid in creating `at_entity_t` structures from C strings.

The `nbp_parse_entity` structure constructs an NBP entity name from a NULL-terminated C string of the form *object, object:type,* or *object:type@zone.* The entity name is placed in the `at_entity_t` structure *entity.* This routine returns 0 on success.

The `nbp_make_entity` structure constructs an NBP entity name from *object, type,* and *zone* strings. The strings are NULL-terminated C strings. The entity name is placed into the `at_entity_t` structure *entity.* Use the *object, type,* and *zone* character strings to construct the entity name. This routine returns 0 on success.

The `nbp_confirm` structure sends a confirmation request to the specified node to see if an entity name is still registered at the specified AppleTalk internet address.

entity A pointer to the `at_entity_t` structure containing the entity name. No wildcards are allowed in the entity name strings, but an asterisk (*) for zone is acceptable.

dest The AppleTalk internet address to confirm. If the name is still registered on the node but at a different socket number, the socket number in *dest* is updated.

retry A pointer to the structure that specifies the NBP request retry interval in seconds and the maximum retry count. If *retry* is NULL, the system uses the default values: a 1-second interval and eight retries.

On success, `nbp_confirm` returns 1. It returns 0 when the name is not confirmed, and –1 on error.

The `nbp_lookup` structure returns a list of registered name-address pairs via an NBP lookup. The parameters are

entity A pointer to the `at_entity_t` structure containing the entity name to be looked up.

buf — An array of `at_nbptuple_t` to receive entity tuples.

max — The maximum number of entity tuples to accept. If *max* or more distinct tuples are received before the lookup retry is exceeded, the lookup terminates.

retry — The pointer to the structure that specifies the NBP request retry interval in seconds and the maximum retry count. If *retry* is NULL, the system uses the default values: a one-second interval and eight retries.

On success, `nbp_lookup` returns the number of entity tuples actually received.

The `nbp_register` structure adds the specified name-socket pair to the list of registered names on this node. The parameters are

entity — A pointer to the `at_entity_t` structure containing the entity name to be registered. The *zone* field of *entity* is always ignored. No wildcards are allowed in the entity strings.

fd — An AppleTalk file descriptor to be registered with the given name.

retry — A pointer to the structure that specifies the NBP request retry interval in seconds and the maximum retry count. If *retry* is NULL, the system uses the default values: a 1-second interval and eight retries.

The `nbp_remove` structure removes the specified entity name from the list of registered names on this node. The parameters are

entity — A pointer to the `at_entity_t` structure containing the entity name to be removed. The *zone* field of *entity* is always ignored. No wildcards are allowed in the entity strings.

fd — The AppleTalk file descriptor that is registered with the given name.

WARNINGS

Strings in entity names and entity tuples are not NULL terminated.

All characters in NVE names are significant, including trailing blanks.

See *Inside AppleTalk* for a description of NVE names.

DIAGNOSTICS

All routines return −1 on error with a detailed error code in *errno:*

[EINVAL]	The entity name is invalid.
[ETIMEDOUT]	The request exceeded maximum retry count.

SEE ALSO

ddp(3n), *Inside AppleTalk.*

NAME

`nlist` — get entries from name list

SYNOPSIS

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

int nlist (filename, nl)
char *filename;
struct nlist *nl;
```

DESCRIPTION

`nlist` examines the name list in the executable file whose name is pointed to by *filename*; it selectively extracts a list of values and puts them in the array of `nlist` structures pointed to by *nl*. The name list *nl* consists of an array of structures containing names of variables, types, and values. The list is terminated with a null name; i.e., a null string is in the name position of the structure. Each variable name is looked up in the name list of the file. If the name is found, the type and value of the name are inserted in the next two fields. The type filed will be set to 0 unless the file was compiled with the `-g` option. If the name is not found, both entries are set to 0. See `a.out`(4) for a discussion of the symbol table structure.

This function is useful for examining the system name list kept in the file `/unix`. In this way programs can obtain system addresses that are up to date.

RETURN VALUE

`nlist` returns –1 upon error; otherwise it returns 0.

All value entries are set to 0 if the file cannot be read or if it does not contain a valid name list.

SEE ALSO

`a.out`(4).

NAME

paps_open, paps_get_next_job, paps_status, paps_close, pap_open, pap_read, pap_read_ignore, pap_status, pap_write, pap_close — AppleTalk Printer Access Protocol (PAP) interface

SYNOPSIS

```
#include <at/appletalk.h>
#include <at/pap.h>
#include <at/nbp.h>
cc [flags] files -lat [libraries]

int paps_open()

int paps_get_next_job(fd)
int fd;

int paps_status(fd, status)
int fd;
char *status;

int paps_close(fd)
int fd;

int pap_open(tuple)
at_nbptuple_t *tuple;

int pap_read(fd, data, len)
int fd, len;
char *data;

int pap_read_ignore(fd)
int fd;

char *pap_status(tuple)
at_nbptuple_t *tuple;

int pap_write(fd, data, len, eof, flush)
int fd, len;
int eof, flush;
char *data;

int pap_close(fd)
int fd;
```

DESCRIPTION

The PAP interface provides applications with access to the AppleTalk Printer Access Protocol operations. The interface routines can be divided into two sets: One set provides services for a PAP client, the other for a PAP server. The routines for the PAP server are

```
paps_open
pap_read
paps_get_next_job
paps_status
paps_close
```

The routines for the PAP client are:

```
pap_open
pap_read
pap_read_ignore
pap_status
pap_write
pap_close
```

The `paps_open` routine opens a PAP server AppleTalk file descriptor for a PAP server. The caller may then use `nbp_register` (see nbp(3N)) to register a network-visible entity (NVE) on the socket and `paps_status` to post a status string on it. The `paps_open` routine returns an AppleTalk file descriptor on success, −1 on failure.

The `paps_get_next_job` routine is called by a server when it is ready to respond to a new PAP client. It returns a PAP server AppleTalk file descriptor that is set up for PAP reading from the client that has been waiting the longest. The parameter is

fd A PAP server AppleTalk file descriptor from a previous `paps_open`.

Upon successful completion a PAP server AppleTalk file descriptor is returned.

The `paps_status` routine changes the `status` string associated with an open PAP server AppleTalk file descriptor. This is the string returned to a PAP client from a `pap_status` call. The parameters are *fd* An open PAP server AppleTalk file descriptor returned from a paps_open call.

status A pointer to a null-terminated character string containing the status string being posted. Strings longer than 255 characters are truncated.

Upon successful completion a value of 0 is returned.

The paps_close routine closes an open PAP server file descriptor. The parameter is

fd The file descriptor to be closed.

It returns 0 upon successful completion.

The pap_open routine opens a PAP client file descriptor to a server. It attempts to connect to the server whose name and address are contained in the *tuple* parameter. The command nbp_lookup (see nbp(3N)) may be used to obtain a valid name and address for the desired PAP server.

Upon successful completion, this routine returns a PAP client file descriptor connected to the server requested.

The pap_read routine reads data from a server PAP file descriptor opened by a paps_open, followed by a paps_get_next_job call. The parameters are

fd A PAP server file descriptor.

data A pointer to the buffer containing the data to be returned. The maximum data length specified by the length parameter is 512 bytes.

length The maximum length to be read.

Upon successful completion, the number of bytes read is returned. A value of 0 is returned when an end-of-file is reached.

The pap_read_ignore routine issues a PAP read request and ignores any returned data. This is used to allow LaserWriters to function when they want to return status messages. The parameter is

fd A PAP client file descriptor returned by an earlier pap_open.

The pap_status routine locates a PAP server and returns a pointer to its status string. The parameter is *tuple* A pointer to a tuple structure containing the name and address of a PAP server entity. The routine nbp_lookup (See nbp(3N)) may be used to get a valid tuple.

Upon successful completion, a pointer to the string containing the PAP server's status is returned. If the printer's status cannot be recovered, NULL is returned.

The pap_write routine sends the data passed to it to the other end of a PAP server session. The parameters are

fd A PAP client AppleTalk file descriptor.

data A pointer to the data being written.

len The length of the data being written; this must not exceed 512 bytes.

eof A Boolean flag indicating whethere EOF indication is to be sent to the other end of the PAP session (after the data has been sent) to indicate that no more data will be sent. Setting *eof* to true also implies *flush*.

flush A Boolean flag indicating whether data for all waiting PAP writes is to be sent to the remote end. Because PAP runs on top of ATP, PAP writes are queued until either a complete ATP response is available (about 4 KB) or an end-of-message is sent. This call sends an ATP end-of-message, which causes all waiting PAP writes to be sent to the other end. This should be done if a higher level protocol (for example, a handshake with a LaserWriter) needs to do a write followed by a read.

Upon successful completion, a value of 0 is returned.

The pap_close routine closes an open PAP client file descriptor. The parameter is

fd The file descriptor to be closed.

It returns 0 upon successful completion. If the file descriptor is no longer open, it returns −1.

ERRORS

All routines except pap_status return −1 on error with a detailed error code in errno:

[EINVAL] An invalid argument was passed.

[ENETDOWN] The network interface is down.

[ESHUTDOWN] The PAP file descriptor has already been closed.

[ETIMEDOUT] The connection is timed out.

See open(2), close(2), ioctl(2), read(2), and write(2) for additional error codes; see also errors returned by the underlying NBP, ATP, and DDP modules.

SEE ALSO

atp(3N), ddp(3N), nbp(3N), rtmp(3N), *Inside AppleTalk.*

NAME

pathconf, fpathconf — get configurable pathname variables

SYNOPSIS

```
#include <unistd.h>

long pathconf(path, name)
char *path;
int name;

long fpathconf(fildes, name)
int fildes, name;
```

DESCRIPTION

pathconf and fpathconf provide a method for an application to determine the current value of a configurable limit or option that is associated with a file or directory.

For fpathconf, *path* points to a pathname of a file or directory. For fpathconf, *fildes* is an open file descriptor. *name* is the variable to be queried relative to the file or directory. The following variables can be queried:

```
_PC_LINK_MAX
_PC_MAX_CANON
_PC_MAX_INPUT
_PC_NAME_MAX
_PC_PATH_MAX
_PC_PIPE_BUF
_PC_CHOWN_RESTRICTED
_PC_CHOWN_SUP_GRP
_PC_DIR_DOTS
_PC_GROUP_PARENT
_PC_LINK_DIR
_PC_NO_TRUNC
_PC_UTIME_OWNER
_PC_VDISABLE
```

RETURN VALUE

If the named variable is not defined on the system, or if *name* is not a valid variable name, or if the variable cannot be associated with the specified file or directory, or if the process does not have permission to query the file specified by *path*, or if *path* does not exist, pathconf returns −1.

If the named variable is not defined on the system, or if *name* is not a valid variable name, or if the variable cannot be associated with the specified file or directory, `fpathconf` returns −1.

If none of the above are true, `pathconf` and `fpathconf` return the current value associated with the variable for the file or directory.

ERRORS

`pathconf` and `fpathconf` will fail if one or more of the following are true:

`[ENOTDIR]`	A component of the path prefix is not a directory.
`[ENAMETOOLONG]`	A component of a pathname exceeded `NAME_MAX` characters, or an entire pathname exceeded `PATH_MAX`.
`[ELOOP]`	Too many symbolic links were encountered in translating a pathname.
`[ENOENT]`	The named file does not exist.
`[EACCES]`	Search permission is denied for a component of the path prefix.
`[EFAULT]`	*path* points to an invalid address.
`[EINVAL]`	The value of the name is invalid, or the variable name is not associated with the specified file.

`fpathconf` will also fail if the following condition occurs:

`[EBADF]`	The open file descriptor, *fildes*, is not valid.

SEE ALSO

`sysconf`(3P).

NAME

perror, errno, sys_errlist, sys_nerr — system error messages

SYNOPSIS

```
void perror(s)
char *s;

extern int errno;

extern char *sys_errlist[];

extern int sys_nerr;
```

DESCRIPTION

perror produces a message on the standard error output, describing the last error encountered during a call to a system or library function. The argument string *s* is printed first, then a colon and a blank, then the message and a newline. To be of most use, the argument string should include the name of the program that incurred the error. The error number is taken from the external variable errno, which is set when errors occur but not cleared when nonerroneous calls are made.

To simplify variant formatting of messages, the array of message strings sys_errlist is provided; errno can be used as an index in this table to get the message string without the newline. sys_nerr is the largest message number provided for in the table; it should be checked because new error codes may be added to the system before they are added to the table.

SEE ALSO

intro(2).

NAME

plot — graphics interface subroutines

SYNOPSIS

```
int openpl()

int erase()

int label(s)
char *s;

int line(x1, y1, x2, y2)
int x1, y1, x2, y2;

int circle(x, y, r)
int x, y, r;

int arc(x, y, x0, y0, x1, y1)
int x, y, x0, y0, x1, y1;

int move(x, y)
int x, y;

int cont(x, y)
int x, y;

int point(x, y)
int x, y;

int linemod(s)
char *s;

int space(x0, y0, x1, y1)
int x0, y0, x1, y1;

int closepl()
```

DESCRIPTION

These subroutines generate graphic output in a relatively device-independent manner. `space` must be used before any of these functions to declare the amount of space necessary; see `plot`(4). `openpl` must be used before any of the others to open the device for writing. `closepl` flushes the output.

`circle` draws a circle of radius *r* with center at the point (*x*,*y*).

`arc` draws an arc of a circle with center at the point (*x*,*y*) between the points (*x0*,*y0*) and (*x1*,*y1*).

String arguments to `label` and `linemod` are terminated by nulls and do not contain newlines.

See plot(4) for a description of the effect of the remaining functions.

The library files listed below provide several variations of these routines.

FILES

/usr/lib/libplot.a	produces output for tplot(1G) filters
/usr/lib/lib300.a	for DASI 300
/usr/lib/lib300s.a	for DASI 300s
/usr/lib/lib450.a	for DASI 450
/usr/lib/lib4014.a	for Tektronix 4014

WARNINGS

To compile a program containing these functions in file.c, use

```
cc file.c -lplot
```

To execute it, use

```
a.out | tplo
```

The above routines use <stdio.h>. Therefore, the size of programs not otherwise using standard I/O is increased more than might be expected.

SEE ALSO

tplot(1G), plot(4).

NAME

popen, pclose — initiate pipe to/from a process

SYNOPSIS

```
#include <stdio.h>

FILE *popen(command, type)
char *command, *type;

int pclose(stream)
FILE *stream;
```

DESCRIPTION

The arguments to popen are pointers to null-terminated strings; one string contains a shell command line and the other contains an I/O mode. The mode may be either "r" for reading or "w" for writing. popen creates a pipe between the calling program and the command to be executed. The value returned is a stream pointer. If the I/O mode is w, one can write to the standard input of the command by writing to the file *stream*; if the I/O mode is r, one can read from the standard output of the command, by reading from the file *stream*.

A stream opened by popen should be closed by pclose, which waits for the associated process to terminate and returns the exit status of the command.

Because open files are shared, a type "r" command may be used as an input filter and a type "w" as an output filter.

RETURN VALUE

popen returns a NULL pointer if files or processes cannot be created.

pclose returns −1 if *stream* is not associated with a command opened by popen.

SEE ALSO

pipe(2), wait(2), fclose(3S), fopen(3S), system(3S).

BUGS

If the original processes and processes opened by popen concurrently read or write a common file, neither should use buffered I/O, because the buffering gets all mixed up. Problems with an output filter may be forestalled by careful buffer flushing, for example, by using fflush (see fclose(3S)).

If an illegal type is passed, popen will fork and exec the command line passed to it before it discovers that the type was illegal. This will result in a NULL pointer being returned and a broken pipe (with the command executing in the background).

NAME

`printf`, `fprintf`, `sprintf` — format and output string and numeric data

SYNOPSIS

```
#include &lt;stdio.h&gt;
```

`int printf(`*format*`[,`*arg*`]...)`
`char *`*format*`;`

`int fprintf(`*stream*`,`*format*`[,`*arg*`]...)`
`FILE *`*stream*`;`
`char *`*format*`;`

`int sprintf(`*s*`,`*format*`[,`*arg*`]...)`
`char *`*s*`,` *format*`;`

DESCRIPTION

`printf` places output on the standard output stream `stdout`. `fprintf` places output on the named output *stream*. `sprintf` places output, followed by the null character (`\0`) in consecutive bytes starting at `*`*s*; it is the user's responsibility to ensure that enough storage is available.

Each of these functions converts, formats, and prints its *arg*s under control of the *format*. The *format* is a character string that contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which results in fetching zero or more *arg*s. The results are undefined if there are insufficient *arg*s for the format. If the format is exhausted while *arg*s remain, the excess *arg*s are simply ignored.

Each conversion specification is introduced by the character `%`. After the `%`, the following appear in sequence:

> Zero or more *flags*, which modify the meaning of the conversion specification.
>
> An optional decimal digit string specifying a minimum *field width*. If the converted value has fewer characters than the field width, it will be padded to the field width on the left (default) or right (if the left-adjustment flag - has been given); see below for flag specification. If the field width for an s conversion is preceded by a 0, the string is right adjusted with zero padding on the left.

A *precision* that gives the minimum number of digits to appear for the d, o, u, x, or X conversions, the number of digits to appear after the decimal point for the e and f conversions, the maximum number of significant digits for the g conversion, or the maximum number of characters to be printed from a string in the s conversion. The format of the precision is a period (.) followed by a decimal digit string; a null digit string is treated as zero.

An optional l (ell) specifying that a following d, o, u, x, or X conversion character applies to a long integer *arg*. An l before any other conversion character is ignored.

A character that indicates the type of conversion to be applied.

A field width or precision may be indicated by an asterisk (*) instead of a digit string. In this case, an integer *arg* supplies the field width or precision. The *arg* that is actually converted is not fetched until the conversion letter is seen; therefore, the *arg*s specifying field width or precision must appear *before* the *arg* (if any) to be converted.

The flag characters and their meanings are:

- `-` The result of the conversion will be left-justified within the field.
- `+` The result of a signed conversion will always begin with a sign (+ or –).
- blank If the first character of a signed conversion is not a sign, a blank will be prefixed to the result. This implies that if the blank and + flags both appear, the blank flag will be ignored.
- `#` This flag specifies that the value is to be converted to an "alternate form." For c, d, s, and u conversions, the flag has no effect. For o conversion, it increases the precision to force the first digit of the result to be a zero. For x (X) conversion, a non-zero result will have 0x (0X) prefixed to it. For e, E, f, g, and G conversions, the result will always contain a decimal point, even if no digits follow the point (normally, a decimal point appears in the result of these conversions only if a digit follows it). For g and G conver-

sions, trailing zeroes will *not* be removed from the result (which they normally are).

The conversion characters and their meanings are:

d,o,u,x,X The integer *arg* is converted to signed decimal, unsigned octal, decimal, or hexadecimal notation (x and X), respectively; the letters abcdef are used for x conversion and the letters ABCDEF for X conversion. The precision specifies the minimum number of digits to appear; if the value being converted can be represented in fewer digits, it will be expanded with leading zeroes. (For compatibility with older versions, padding with leading zeroes may alternatively be specified by prefixing a zero to the field width.) This does not imply an octal value for the field width. The default precision is 1. The result of converting a zero value with a precision of zero is a null string.

f The float or double *arg* is converted to decimal notation in the style [-]*ddd*.*ddd*, where the number of digits after the decimal point is equal to the precision specification. If the precision is missing, 6 digits are output; if the precision is explicitly 0, no decimal point appears.

e,E The float or double *arg* is converted in the style [-]*d*.*ddd*e±*dd*, where there is one digit before the decimal point and the number of digits after it is equal to the precision; when the precision is missing, 6 digits are produced; if the precision is zero, no decimal point appears. The E format code produces a number with E instead of e introducing the exponent. The exponent always contains at least two digits.

g,G The float or double *arg* is printed in style f or e (or in style E in the case of a G format code), with the precision specifying the number of significant digits. The style used depends on the value converted: style e is used only if the exponent resulting from the conversion is less than –4 or greater than the precision. Trailing zeroes are removed from the result; a decimal point appears only if it is followed by a digit.

c The character *arg* is printed.

s The *arg* is taken to be a string (character pointer) and characters from the string are printed until a null character (\0) is encountered or the number of characters indicated by the precision specification is reached. If the precision is missing, it is taken to be infinite, so all characters up to the first null character are printed. A NULL value for *arg* yields undefined results.

% Print a %; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; if the result of a conversion is wider than the field width, the field is simply expanded to contain the conversion result. Characters generated by `printf` and `fprintf` are printed as if `putc`(3S) had been called.

RETURN VALUE

Each function returns the number of characters transmitted (not including the \0 in the case of `sprintf`), or a negative value if an output error was encountered.

EXAMPLES

To print a date and time in the form "Sunday, July 3, 10:02", where *wkday* and *mnth* are pointers to null-terminated strings:

```
printf("%s, %s %d, %.2d:%.2d", wkday, mnth, day, hr, min);
```

To print *pi* to 5 decimal places:

```
printf("pi=%.5f", 4*atan(1.0));
```

SEE ALSO

`ecvt`(3C), `intro`(3), `putc`(3S), `scanf`(3S).

NAME

`putc`, `putchar`, `fputc`, `putw` — put character or word on a stream

SYNOPSIS

```
#include &lt;stdio.h&gt;

int putc(c, stream)
int c;
FILE *stream;

int putchar(c)
int c;

int fputc(c, stream)
int c;
FILE *stream;

int putw(w, stream)
int w;
FILE *stream;
```

DESCRIPTION

The `putc` macro writes the character *c* onto the output *stream* at the position where the file pointer, if defined, is pointing. The `putchar` macro is defined as `putc(`*c*`,` *stdout*`)`.

`fputc` behaves like `putc`, but is a function rather than a macro. `fputc` runs more slowly than `putc`, but it takes less space per invocation and its name can be passed as an argument to a function.

`putw` writes the word (32-bit integer on the Macintosh II) *w* to the output *stream* at the position at which the file pointer, if defined, is pointing. `putw` neither assumes nor causes special alignment in the file.

Output streams, with the exception of the standard error stream `stderr`, are by default buffered if the output refers to a file and line-buffered if the output refers to a terminal. The standard error output stream `stderr` is by default unbuffered, but use of `freopen` (see `fopen`(3S)) causes it to become buffered or line-buffered. When an output stream is unbuffered information, it is queued for writing on the destination file or terminal as soon as written; when it is buffered, many characters are saved up and written as a block; when it is line-buffered, each line of output is queued for writing on the destination terminal as soon as the line is completed (i.e., as soon as a newline character is written or terminal input is requested). setbuf(3S) may be used to change the stream's buffering strategy.

RETURN VALUE

On success, these functions each return the value they have written. On failure, they return the constant EOF. This occurs if the file *stream* is not open for writing or if the output file cannot be grown. Because EOF is a valid integer, ferror(3S) should be used to detect putw errors.

SEE ALSO

fclose(3S), ferror(3S), fopen(3S), fread(3S), getc(3S), printf(3S), puts(3S), setbuf(3S).

BUGS

Because it is implemented as a macro, putc treats incorrectly a *stream* argument with side effects. In particular, `putc(c, *f++);` doesn't work sensibly. fputc should be used instead. Because of possible differences in word length and byte ordering, files written using putw are machine-dependent and may not be read using getw on a different processor.

NAME

putenv — change or add value to environment

SYNOPSIS

```
int putenv(string)
char *string;
```

DESCRIPTION

string points to a string of the form "*name=value*". putenv makes the value of the environment variable *name* equal to *value* by altering an existing variable or creating a new one. In either case, the string pointed to by *string* becomes part of the environment, so altering the string will change the environment. The space used by *string* is no longer used once a new string-defining *name* is passed to putenv.

RETURN VALUE

putenv returns nonzero if it was unable to obtain enough space via malloc for an expanded environment, otherwise zero.

SEE ALSO

exec(2), getenv(3C), malloc(3C), environ(5).

WARNINGS

putenv manipulates the environment pointed to by environ, and can be used in conjunction with getenv. However, *envp* (the third argument to *main*) is not changed.

This routine uses malloc(3C) to enlarge the environment.

After putenv is called, environmental variables are not in alphabetical order.

A potential error is to call putenv with an automatic variable as the argument, then exit the calling function while *string* is still part of the environment.

NAME

putpwent — write password file entry

SYNOPSIS

```
#include <pwd.h>

int putpwent(p, f)
struct passwd *p;
FILE *f;
```

DESCRIPTION

putpwent is the inverse of getpwent(3C). Given a pointer to a passwd structure created by getpwent (or getpwuid or getpwnam), putpwuid writes a line on the stream *f* which matches the format of /etc/passwd.

The <pwd.h> header file is described in getpwent(3C).

RETURN VALUE

putpwent returns nonzero if an error was detected during its operation; otherwise it returns zero.

SEE ALSO

getpwent(3C).

WARNINGS

The above routine uses <stdio.h>. Therefore, the size of programs not otherwise using standard I/O is increased more than might be expected.

NAME

`puts`, `fputs` — put a string on a stream

SYNOPSIS

```
#include <stdio.h>

int puts(s)
char *s;

int fputs(s, stream)
char *s;
FILE *stream;
```

DESCRIPTION

`puts` writes the null-terminated string referenced by *s*, followed by a newline character, to the standard output stream `stdout`.

`fputs` writes the null-terminated string pointed to by *s* to the named output *stream*.

Neither function writes the terminating null character.

SEE ALSO

`ferror`(3S), `fopen`(3S), `fread`(3S), `printf`(3S), `putc`(3S).

RETURN VALUE

On success, both routines return the number of characters written.

Both functions return EOF on error. This occurs if the routines try to write on a file that has not been opened for writing.

NOTES

`puts` appends a newline character while `fputs` does not.

NAME

qsort — quicker sort

SYNOPSIS

```
void qsort(base, nel, width, compar)
char *base;
unsigned nel, width;
int(*compar)();
```

DESCRIPTION

qsort is an implementation of the quicker-sort algorithm. It sorts a table of data in place.

base points to the element at the base of the table. *nel* is the number of elements in the table. *width* is the width of an element in bytes. *compar* is the name of the comparison function, which is called with two arguments that point to the elements being compared. The function must return an integer less than, equal to, or greater than zero according as the first argument is to be considered less than, equal to, or greater than the second.

NOTES

The pointer to the base of the table should be of type pointer-to-element, and cast to type pointer-to-character.

The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared. The order in the output of the two items which compare as equal is unpredictable.

EXAMPLES

```
struct  entry {
        char    *name;
        int     flags;
};

main()
{
        struct entry hp[100];
        int entcmp();
        int i, count;

        for (i = 0; i < (count = 100); i++) {
                /* fill the structure with the name
                   and flags */
        ...
        }
        qsort( (char *) hp, count, sizeof (hp[0]), entcmp);
}
```

```
entcmp(ep,ep2)
struct entry *ep, *ep2;
{
        return (strcmp(ep->name, ep2->name));
}
```

will sort a set of names with associated flags in ASCII order.

SEE ALSO

sort(1), bsearch(3C), lsearch(3C), string(3C).

NAME

`rand`, `srand` — simple random-number generator

SYNOPSIS

```
int rand()
```

```
void srand(seed)
unsigned seed;
```

DESCRIPTION

`rand` uses a multiplicative congruential random-number generator with period 2 power of 32 that returns successive pseudo-random numbers in the range from 0 to 32767.

`srand` can be called at any time to reset the random-number generator to a random starting point. The generator is initially seeded with a value of 1.

NOTES

The spectral properties of `rand` leave a great deal to be desired. `drand48`(3C) provides a much better, though more elaborate, random-number generator.

SEE ALSO

`drand48`(3C).

NAME

irand, srand, rand — Fortran uniform random-number generator

SYNOPSIS

call srand(*iseed*)

i=irand()

x=rand()

DESCRIPTION

irand generates successive pseudo-random numbers in the range from 0 to 2**15-1. rand generates pseudo-random numbers distributed in (0, 1.0). srand uses its integer argument to reinitialize the seed for successive invocations of irand and rand.

SEE ALSO

rand(3C).

NAME

rcmd, rresvport, ruserok — routines for returning a stream to a remote command

SYNOPSIS

```
int rcmd(ahost, inport, locuser, remuser, cmd, fd2p)
char **ahost;
u_short inport;
char *locuser, *remuser, *cmd;
int *fd2p;

int rresvport(port)
int *port;

int ruserok(rhost, superuser, user, user)
char *rhost;
int superuser;
char *ruser, *luser;
```

DESCRIPTION

rcmd is a routine used by the superuser to execute a command on a remote machine using an authentication scheme based on reserved port numbers. rresvport is a routine which returns a descriptor to a socket with an address in the privileged port space. ruserok is a routine used by servers to authenticate clients requesting service with rcmd. All three functions are present in the same file and are used by the remshd(1M) server, as well as others.

rcmd looks up the host **ahost*, returning −1 if the host does not exist. Otherwise **ahost* is set to the standard name of the host and a connection is established to a server residing at the well-known Internet port *inport*.

If the call succeeds, a socket of type SOCK_STREAM is returned to the caller, and given to the remote command as stdin and stdout. If *fd2p* is nonzero, then an auxiliary channel to a control process will be set up, and a descriptor for it will be placed in **fd2p*. The control process will return the stderr (descriptor 2 of the remote(1M) command) on this channel and will accept bytes on this channel as A/UX signal numbers to be forwarded to the process group of the command. If *fd2p* is 0, then the stderr (descriptor 2 of the remote(1M) command) will be made the same as stdout; no provision will be made for sending arbitrary signals to the remote process, although you may be able to get its attention by using out-of-band data.

The protocol is described in detail in remshd(1M).

The rresvport routine is used to obtain a socket with a privileged address bound to it. This socket is suitable for use by rcmd and several other routines. Privileged addresses consist of a port in the range 0 to 1023. Only the superuser is allowed to bind an address of this sort to a socket.

ruserok takes a remote host's name, two user names, and a flag indicating if the local user's name is the superuser. It then checks the files /etc/hosts.equiv and, possibly, .rhosts in the current working directory (normally the local user's home directory) to see if the request for service is allowed. A 0 is returned if the machine name is listed in the hosts.equiv file or the host and remote user name are found in the .rhosts file; otherwise ruserok returns −1. If the *superuser* flag is 1, the checking of the host.equiv file is bypassed.

SEE ALSO

remsh(1N), rlogin(1N), remshd(1M), rexecd(1M), rlogind(1M), rexec(3N).

BUGS

There is no way to specify options to the socket call which rcmd makes.

NAME

regcmp, regex — compile and execute a regular expression

SYNOPSIS

char *regcmp(*string1* [, *string2*, ...], (char *)0))
char **string1*, **string2*, ...;

char *regex(*re*, *subject* [, *ret0*, ...])
char **re*, **subject*, **ret0*, ...;

extern char **loc1*;

DESCRIPTION

regcmp compiles a regular expression and returns a pointer to the compiled form. malloc(3C) is used to create space for the vector. It is the user's responsibility to free unneeded space that has been allocated by malloc. A NULL return from regcmp indicates an incorrect argument. regcmp(1) has been written to generally preclude the need for this routine at execution time.

regex executes a compiled pattern against the subject string. Additional arguments are passed to receive values back. regex returns NULL on failure or a pointer to the next unmatched character on success. A global character pointer loc1 points to where the match began. regcmp and regex were mostly borrowed from the editor, ed(1); however, the syntax and semantics have been changed slightly. The following are the valid symbols and their associated meanings.

[ ] * . ^ These symbols retain their current meaning.

$ This symbol matches the end of the string; \n matches the newline.

– Within brackets the minus means "through." For example, [a–z] is equivalent to [abcd. . .xyz]. The – can appear as itself only if used as the last or first character. For example, the character class expression []–] matches the characters] and –.

\+ A regular expression followed by + means "one or more times." For example, [0–9]+ is equivalent to [0–9][0–9]*.

{*m*} {*m*,} {*m*,*u*} Integer values enclosed in { } indicate the number of times the preceding regular expression is to be applied. The minimum number is *m* and the maximum number is *u*, which must be less than 256.

If only *m* is present (e.g., `{m}`), it indicates the exact number of times the regular expression is to be applied. `{m,}` is analogous to `{m,infinity}`. The plus (+) and star (*) operations are equivalent to `{1,}` and `{0,}`, respectively.

`( ... )$n`
: The value of the enclosed regular expression is to be returned. The value will be stored in the *(n+1)*th argument following the subject argument. At present, at most 10 enclosed regular expressions are allowed. `regex` makes its assignments unconditionally.

`( ... )`
: Parentheses are used for grouping. An operator (e.g., `*`, `+`, `{ }`) can work on a single character or a regular expression enclosed in parentheses. For example, (a*(cb+)*)$0.

By necessity, all the above defined symbols are special. They must, therefore, be escaped to be used as themselves.

EXAMPLES

Example 1:

```
char *cursor, *newcursor, *ptr;
       ...
newcursor = regex((ptr = regcmp("^\n", 0)), cursor);
free(ptr);
```

This example will match a leading newline in the subject string pointed at by cursor.

Example 2:

```
char ret0[9];
char *newcursor, *name;
       ...
name = regcmp("([A-Za-z][A-za-z0-9_]{0,7})$0", 0);
newcursor = regex(name, "123Testing321", ret0);
```

This example will match through the string "`Testing3`" and will return the address of the character after the last matched character (cursor+11). The string "`Testing3`" will be copied to the character array `ret0`.

Example 3:

```
#include "file.i"
char *string, *newcursor;
        . . .
newcursor = regex(name, string);
```

This example applies a precompiled regular expression in `file.i` (see `regcmp`(1)) against *string*.

This routine is kept in `/lib/libPW.a`.

SEE ALSO

`ed`(1), `regcmp`(1), `malloc`(3C).

BUGS

The user program may run out of memory if `regcmp` is called iteratively without freeing the vectors no longer required. The following user-supplied replacement for `malloc`(3C) reuses the same vector, saving time and space:

```
/* user's program */

char *
malloc(n)
unsigned n;
{
        static char rebuf[512];
        return (n <= sizeof rebuf) ? rebuf : NULL;
}
```

NAME

res_mkquery, res_send, res_init, dn_comp, dn_expand — resolver routines

SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>
#include <arpa/nameser.h>
#include <resolv.h>

res_mkquery(op, dname, class, type, data, datalen,
        newrr, buf, buflen)
int op;
char *dname;
int class, type;
char *data;
int datalen;
struct rrec *newrr;
char *buf;
int buflen;

res_send(msg, msglen, answer, anslen)
char *msg;
int msglen;
char *answer;
int anslen;

res_init()

dn_comp(exp_dn, comp_dn, length, dnptrs, lastdnptr)
char *exp_dn, *comp_dn;
int length;
char **dnptrs, **lastdnptr;

dn_expand(msg, eomorig, comp_dn, exp_dn, length)
char *msg, *eomorig, *comp_dn, exp_dn;
int length;
```

DESCRIPTION

These routines are used for making, sending, and interpreting packets to Internet domain name servers. Global information that is used by the `resolver` routines is kept in the variable `_res`. Most of the values have reasonable defaults and can be ignored. Options stored in `_res.options` are defined in `resolv.h` and are as follows. Options are a simple bit mask.

RES_INIT
: True if the initial name server address and default domain name are initialized (for example, res_init has been called).

RES_DEBUG
: Print debugging messages.

RES_AAONLY
: Accept authoritative answers only. res_send will continue until it finds an authoritative answer or finds an error. Currently this is not implemented.

RES_USEVC
: Use TCP connections for queries instead of UDP.

RES_STAYOPEN
: Used with RES_USEVC to keep the TCP connection open between queries. This is useful only in programs that regularly do many queries. UDP should be the normal mode used.

RES_IGNTC
: Unused currently (ignore truncation errors—don't retry with TCP).

RES_RECURSE
: Set the recursion desired bit in queries. This is the default. (res_send does not do iterative queries and expects the name server to handle recursion.)

RES_DEFNAMES
: Append the default domain name to single label queries. This is the default.

res_init

reads the initialization file to get the default domain name and the Internet address of the initial hosts running the name server. If this line does not exist, the host running the resolver is tried. res_mkquery makes a standard query message and places it in *buf*. res_mkquery will return the size of the query or −1 if the query is larger than *buflen*. *op* is usually QUERY but can be any of the query types defined in nameser.h. *dname* is the domain name. If *dname* consists of a single label and the RES_DEFNAMES flag is enabled (the default), *dname* will be appended with the current domain name. The current domain name is defined in a system file and can be overridden by the environment variable LOCALDOMAIN. *newrr* is currently unused but is intended for making update messages.

res_send sends a query to name servers and returns an answer. It will call res_init if RES_INIT is not set, send the query to the local name server, and handle timeouts and retries. The length of the message is returned or −1 if there were errors.

dn_expand expands the compressed domain name comp_dn to a full domain name. Expanded names are converted to uppercase. *msg* is a pointer to the beginning of the message, *exp_dn* is a pointer to a buffer of size *length* for the result. The size of compressed name is returned or −1 if there was an error.

dn_comp compresses the domain name *exp_dn* and stores it in comp_dn. The size of the compressed name is returned or −1 if there were errors. *length* is the size of the array pointed to by comp_dn. *dnptrs* is a list of pointers to previously compressed names in the current message. The first pointer points to to the beginning of the message and the list ends with NULL. *lastdnptr* is a pointer to the end of the array pointed to *dnptrs*. A side effect is to update the list of pointers for labels inserted into the message by *dn_comp* as the name is compressed. If *dnptr* is NULL, we don't try to compress names. If *lastdnptr* is NULL, we don't update the list.

FILES

/etc/resolv.conf

SEE ALSO

named(1M), resolver(4).

NAME

rexec — return stream to a remote command

SYNOPSIS

```
int rexec(ahost, inport, user, passwd, cmd, fd2p);
char **ahost;
u_short inport;
char *user, *passwd, *cmd;
int *fd2p;
```

DESCRIPTION

rexec looks up the host referenced by **ahost* using gethostbyname(3N), returning −1 if the host does not exist. Otherwise **ahost* is set to the standard name of the host. If a username and password are both specified, then these are used to authenticate to the foreign host; otherwise the environment and then the user's .netrc file in his home directory are searched for appropriate information. If all this fails, the user is prompted for the information.

The port *inport* specifies which well-known DARPA Internet port to use for the connection; it will normally be the value returned from the call

```
getservbyname("exec", "tcp")
```

(see getservent(3N)). The protocol for connection is described in detail in rexecd(1M).

If the call succeeds, a socket of type SOCK_STREAM is returned to the caller, and given to the remote command as stdin and stdout. If *fd2p* is nonzero, then a auxiliary channel to a control process will be setup and a descriptor for it will be placed in **fd2p*. The control process will return diagnostic output from the command (unit 2) on this channel, and will also accept bytes on this channel as being A/UX signal numbers, to be forwarded to the process group of the command. If *fd2p* is 0, then the stderr (unit 2 of the remote command) will be made the same as the stdout and no provision is made for sending arbitrary signals to the remote process, although you may be able to get its attention by using out-of-band data.

SEE ALSO

rcmd(3N), rexecd(1M).

BUGS

There is no way to specify options to the `socket` call which `rexec` makes.

NAME

rnusers, rusers — return information about users on remote machines

SYNOPSIS

```
#include <rpcsvc/rusers.h>

rnusers(host)
char *host;

rusers(host,up)
char *host;
struct utmpidlearr *up;
```

DESCRIPTION

rnusers returns the number of users logged on to *host* (or −1 if it cannot determine that number). rusers fills the structure utmpidlearr with data about *host*, and returns 0 if successful. The relevant structures are

```
struct utmparr {     /* RUSERSVERS_ORIG */
   struct utmp **uta_arr;
   int uta_cnt
};

struct utmpidle {
   struct utmp ui_utmp;
   unsigned ui_idle;
};

struct utmpidlearr { /* RUSERSVERS_IDLE */
   struct utmpidle **uia_arr;
   int uia_cnt
};
```

RPC INFO

Program number: RUSERSPROG

xdr routines:

```
int xdr_utmp(xdrs, up)
   XDR *xdrs;
   struct utmp *up;
int xdr_utmpidle(xdrs, ui);
   XDR *xdrs;
   struct utmpidle *ui;
int xdr_utmpptr(xdrs, up);
   XDR *xdrs;
   struct utmp **up;
```

```
int xdr_utmpidleptr(xdrs, up);
    XDR *xdrs;
    struct utmpidle **up;
int xdr_utmparr(xdrs, up);
    XDR *xdrs;
    struct utmparr *up;
int xdr_utmpidlearr(xdrs, up);
    XDR *xdrs;
    struct utmpidlearr *up;
```

Procs:

RUSERSPROC_NUM
: No arguments; returns number of users as an `unsigned long`.

RUSERSPROC_NAMES
: No arguments; returns `utmparr` or `utmpidlearr`, depending on version number.

RUSERSPROC_ALLNAMES
: No arguments; returns `utmparr` or `utmpidlearr`, depending on version number. Returns listing even for `utmp` entries satisfying `nonuser()` in `utmp.h`.

Versions:
RUSERSVERS_ORIG, RUSERSVERS_IDLE

SEE ALSO

rusers(1), rusersd(1M).

NAME

anint, dnint, nint, idnint — Fortran nearest integer functions

SYNOPSIS

```
integer i
real r1, r2
double precision dp1, dp2

r2=anint(r1)
i=nint(r1)

dp2=anint(dp1)
dp2=dnint(dp1)

i=nint(dp1)
i=idnint(dp1)
```

DESCRIPTION

`anint` returns the nearest whole real number to its real argument (i.e., `int(a+0.5)` if `a ≥ 0`, `int(a-0.5)` otherwise). `dnint` does the same for its double-precision argument. `nint` returns the nearest integer to its real argument. `idnint` is the double-precision version. `anint` is the generic form of `anint` and `dnint`, performing the same operation and returning the data type of its argument. `nint` is also the generic form of `idnint`.

NAME

rpc — library routines for remote procedure calls

DESCRIPTION

These routines allow C programs to make procedure calls on other machines across the network. First, the client calls a procedure to send a data packet to the server. Upon receipt of the packet, the server calls a dispatch routine to perform the requested service, and then sends back a reply. Finally, the procedure call returns to the client.

FUNCTIONS

`auth_destroy()`	destroy authentication information handle
`authnone_create()`	return RPC authentication handle with no checking
`authunix_create()`	return RPC authentication handle with A/UX permissions
`authunix_create_default()`	return default A/UX authentication handle
`callrpc()`	call remote procedure, given [*prognum*,*versnum*,*procnum*]
`clnt_broadcast()`	broadcast remote procedure call everywhere
`clnt_call()`	call remote procedure associated with client handle
`clnt_destroy()`	destroy client's RPC handle
`clnt_freeres()`	free data allocated by RPC/XDR system when decoding results
`clnt_geterr()`	copy error information from client handle to error structure
`clnt_pcreateerror()`	print message to stderr about why client handle creation failed

`clnt_perrno()`	print message to stderr corresponing to condition given
`clnt_perror()`	print message to stderr about why RPC call failed
`clnt_sperrno()`	print message to a string corresponding to condition given
`clnt_sperror()`	print message to a string
`clntraw_create()`	create toy RPC client for simulation
`clnttcp_create()`	create RPC client using TCP transport
`clntudp_create()`	create RPC client using UDP transport
`get_myaddress()`	get the machine's IP address
`pmap_getmaps()`	return list of RPC program-to-port mappings
`pmap_getport()`	return port number on which waits supporting service
`pmap_rmtcall()`	instructs portmapper to make an RPC call
`pmap_set()`	establish mapping between [prognum,versnum,procnum] and port
`pmap_unset()`	destroy mapping between [*prognum,versnum,procnum*] and port
`registerrpc()`	register procedure with RPC service package
`rpc_createerr`	global variable indicating reason why client creation failed

`svc_destroy()`	destroy RPC service transport handle
`svc_fds`	global variable with RPC service file descriptor mask
`svc_freeargs()`	free data allocated by RPC/XDR system when decoding arguments
`svc_getargs()`	decodes the arguments of an RPC request
`svc_getcaller()`	get the network address of the caller of a procedure
`svc_getreq()`	returns when all associated sockets have been serviced
`svc_register()`	associates prognum and versnum with service dispatch procedure
`svc_run()`	wait for RPC requests to arrive and call appropriate service
`svc_sendreply()`	send back results of a remote procedure call
`svc_unregister()`	remove mapping of [prognum,versnum] to dispatch routines
`svcerr_auth()`	called when refusing service because of authentication error
`svcerr_decode()`	called when service cannot decode its parameters
`svcerr_noproc()`	called when service hasn't implemented the desired procedure
`svcerr_noprog()`	called when program is not registered with RPC package
`svcerr_progvers()`	called when version is not registered with RPC pack-

	age
`svcerr_systemerr()`	called when service detects system error
`svcerr_weakauth()`	called when refusing service because of insufficient authentication
`svcraw_create()`	creates a toy RPC service transport for testing
`svctcp_create()`	creates an RPC service based on TCP transport
`svcudp_create()`	creates an RPC service based on UDP transport
`xdr_accepted_reply()`	generates RPC-style replies without using RPC package
`xdr_authunix_parms()`	generates A/UX credentials without using RPC package
`xdr_callhdr()`	generates RPC-style headers without using RPC package
`xdr_callmsg()`	generates RPC-style messages without using RPC package
`xdr_opaque_auth()`	describes RPC messages, externally
`xdr_pmap()`	describes parameters for portmap procedures, externally
`xdr_pmaplist()`	describes a list of port mappings, externally
`xdr_rejected_reply()`	generates RPC-style rejections without using RPC package
`xdr_replymsg()`	generates RPC-style replies without using RPC package
`xprt_register()`	registers RPC service transport with RPC package

`xprt_unregister()`	unregisters RPC service transport from RPC package

SEE ALSO

A/UX Network Applications Programming.

NAME

`rtmp_netinfo` — identify AppleTalk node and bridge addresses

SYNOPSIS

```
#include <at/appletalk.h>
cc [flags] files -lat [libraries]
```

```
int rtmp_netinfo(fd, addr, bridge)
int fd;
at_inet_t *addr, *bridge;
```

DESCRIPTION

This routine allows the caller to determine node addresses. It uses the structure `at_inet_t` defined in `<at/appletalk.h>`:

```
typedef struct at_inet {
        at_net          net;
        at_node         node;
        at_socket       socket;
} at_inet_t;
```

The `at_inet_t` structure specifies AppleTalk socket internet address. The parameters are

fd An AppleTalk socket descriptor. If this parameter is −1, it is ignored; otherwise, upon return, the *socket* field in *addr* contains the socket number corresponding to *fd*.

addr Pointer to an `at_inet_t` structure. If this pointer is non-NULL, the AppleTalk network and node addresses are returned in the structure to which it points. If *fd* is not −1, the *socket* field of this structure is filled, otherwise it is zero. This parameter is ignored if it is NULL.

bridge Pointer to an `at_inet_t` structure. If this pointer is non-NULL, the AppleTalk network and addresses of a bridge known to DDP are returned in the structure to which it points. This parameter is ignored if it is NULL. The *socket* field is meaningless and always contains zero on return.

Either *addr* or *bridge* must be non-NULL. rtmp_netinfo returns an error if both are NULL.

The function returns zero if successful; otherwise, −1 is returned with a detailed error code in errno.

DIAGNOSTICS

rtmp_netinfo returns −1 on error with a detailed error code in errno:

[EINVAL]	Both addr and bridge are NULL

See also the errors returned by the underlying DDP module.

SEE ALSO

ddp(3N), *Inside AppleTalk;* "AppleTalk Programming Guide," in *A/UX Network Applications Programming.*

NAME

rwall — write to specified remote machines

SYNOPSIS

```
#include &lt;rpcsvc/rwall.h&gt;

rwall(host, msg);
char *host, *msg;
```

DESCRIPTION

`rwall` causes *host* to print the string *msg* to all its users. It returns 0 if successful.

RPC INFO

Program number: `WALLPROG`

Procs:

`WALLPROC_WALL`
: Takes string as argument (wrapstring); returns no arguments. Executes `wall` on remote host with string.

Versions: `RSTATVERS_ORIG`

SEE ALSO

`rwall`(1M), `shutdown`(8), `rwalld`(1M).

NAME

scandir — scan a directory

SYNOPSIS

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

scandir(dirname, namelist, select, compar)
char *dirname;
struct direct *(*namelist[]);
int (*select)();
int (*compar)();

alphasort(d1, d2)
struct direct **d1, **d2;
```

DESCRIPTION

scandir reads the directory *dirname* and builds an array of pointers to directory entries using malloc(3). It returns the number of entries in the array and a pointer to the array through *namelist.*

The *select* parameter is a pointer to a user supplied subroutine which is called by scandir to select which entries are to be included in the array. The select routine is passed a pointer to a directory entry and should return a non-zero value if the directory entry is to be included in the array. If *select* is null, then all the directory entries will be included.

The *compar* parameter is a pointer to a user supplied subroutine which is passed to qsort(3) to sort the completed array. If this pointer is null, the array is not sorted. alphasort is a routine which can be used for the *compar* parameter to sort the array alphabetically.

The memory allocated for the array can be deallocated with free (see malloc(3)) by freeing each pointer in the array and the array itself.

RETURN VALUE

Returns −1 if the directory cannot be opened for reading or if cannot allocate enough memory to hold all the data structures.

SEE ALSO

directory(3), malloc(3C), malloc(3X), qsort(3C), dir(4).

NAME

scanf, fscanf, sscanf — convert formatted input

SYNOPSIS

```
#include <stdio.h>

int scanf (format [, pointer]... )
char *format;

int fscanf (stream, format [, pointer]... )
FILE *stream;
char *format;

int sscanf (s, format [, pointer]... )
char *s, *format;
```

DESCRIPTION

scanf reads from the standard input stream stdin. fscanf reads from the named input *stream*. sscanf reads from the character string at *s. Each function reads characters, interprets them according to *format*, and stores the results in the location specified by the *pointer* arguments. Each function expects as arguments: a control string *format* (described below) and a set of *pointer* arguments indicating where the converted input should be stored.

The control string usually contains conversion specifications, which are used to direct interpretation of input sequences. The control string may contain:

1. White-space characters (blanks and tabs) which, except in two cases described below, cause input to be read up to the next nonwhite-space character.
2. An ordinary character (not %), which must match the next character of the input stream.
3. Conversion specifications, consisting of the character %, an optional assignment suppression character *, an optional numerical maximum field width, an optional letter l or h indicating the size of the receiving variable, and a conversion code.

A conversion specification directs the conversion of the next input field; the result is placed in the variable pointed to by the corresponding argument, unless assignment suppression has been indicated by *. The suppression of assignment provides a way of describing an input field which is to be skipped. An input field is defined as a string of nonwhite-space characters; it extends to the next inappropriate character or until the field width, if specified, is exhausted. For all descriptors except "[" and "c", white space leading an input field is ignored.

The conversion code indicates the interpretation of the input field; the corresponding pointer argument must usually be of a restricted type. For a suppressed field, no pointer argument should be given. The following conversion codes are legal:

`%` A single `%` is expected in the input at this point; no assignment is done.

`d` A decimal integer is expected; the corresponding argument should be an integer pointer.

`u` An unsigned decimal integer is expected; the corresponding argument should be an unsigned integer pointer.

`o` An octal integer is expected; the corresponding argument should be an integer pointer.

`x` A hexadecimal integer is expected; the corresponding argument should be an integer pointer.

`e,f,g` A floating point number is expected; the next field is converted accordingly and stored through the corresponding argument, which should be a pointer to a *float*. The input format for floating point numbers is an optionally signed string of digits, possibly containing a decimal point, followed by an optional exponent field consisting of an `E` or an `e`, followed by an optional +, –, or space followed by an integer.

`s` A character string is expected; the corresponding argument should be a character pointer to an array of characters large enough to accept the string and a terminating `\0`, which will be added automatically. The input field is terminated by a white-space character.

`c` A character is expected; the corresponding argument should be a character pointer. The normal skip over white space is suppressed in this case; to read the next nonspace character, use `%1s`. If a field width is given, the corresponding argument should refer to a character array; the indicated number of characters is read.

`[` String data and the normal skip over leading white space is suppressed. The left bracket is followed by a set of characters (the *scanset*) and a right bracket; the input field is the maximal sequence of input characters consisting entirely of characters in the *scanset*. The caret, (^), when it appears as the first character in the *scanset*, serves as a complement operator and redefines the *scanset* as the set of all characters *not* contained in the remainder of the *scanset* string. There are some conventions used in the construction of the *scanset*. A range of characters may be represented by the construct *first-last*; thus, `[0123456789]` may be expressed `[0-9]`. Using this convention, *first* must be lexically less than or equal to *last*, or else the dash will stand for itself. The dash will also stand for itself whenever it is the first or the last character in the *scanset*. To include the right square bracket as an element of the *scanset*, it must appear as the first character (possibly preceded by a circumflex) of the *scanset*; otherwise it will be interpreted syntactically as the closing bracket. The corresponding argument must point to a character array large enough to hold the data field and the terminating `\0`, which will be added automatically. At least one character must match for this conversion to be considered successful.

The conversion characters `d`, `u`, `o`, and `x` may be preceded by `l` or `h` to indicate that a pointer to `long` or `short`, rather than `int`, is in the argument list. Similarly, the conversion characters `e`, `f`, and `g` may be preceded by `l` to indicate that a pointer to `double`, rather than `float`, is in the argument list.

The `l` or `h` modifier is ignored for other conversion characters. `scanf` conversion terminates at EOF, at the end of the control string, or when an input character conflicts with the control string. In the latter case, the offending character is left unread in the input stream.

`scanf` returns the number of successfully matched and assigned input items; this number can be zero when an early conflict between an input character and the control string occurs. If the input ends before the first conflict or conversion, EOF is returned.

EXAMPLES

The call:

```
int i; n; float x; char name[50];
n = scanf ("%d%f%s", &i, &x, name);
```

with the input line

```
25 54.32E-1 thompson
```

will assign the value 3 to *n*, the value 25 to *i*, and the value 5.432 to *x*; *name* will contain thompson\0.

The call

```
int i; float x; char name[50];
(void) scanf ("%2d%f%*d %[0-9]", &i, &x,
name);
```

with input

```
56789 0123 56a72
```

will assign 56 to *i*, 789.0 to *x*, skip 0123, and place the string 56\0 in *name*. The next call to getchar (see getc(3S)) will return a.

RETURN VALUE

These functions return EOF on end of input and a short count for missing or illegal data items.

NOTES

Trailing white space is left unread unless matched in the control string.

BUGS

The success of literal matches and suppressed assignments is not directly determinable.

SEE ALSO

getc(3S), printf(3S), strtod(3C), strtol(3C).

NAME

set42sig — set 4.2 BSD signal interface

SYNOPSIS

```
int set42sig()
```

DESCRIPTION

set42sig changes the signal interface to one closely resembling BSD 4.2 systems. This call is similar to the setcompat system call. Unlike setcompat(2), set42sig arranges for the current compatibility flags to be logically OR'ed with the new flags. set42sig is functionally equivalent to the following C code fragment:

```
#include <compat.h>

return (setcompat(getcompat() | COMPAT_BSDSIGNALS |
        COMPAT_BSDTTY | COMPAT_BSDSYSCALLS));
```

For the process calling it, it enables reliable signal delivery, the job control tty signals, and restarting of system calls when an interrupt is received.

If the COMPAT_SVID flag is set before calling set42sig, both BSD 4.2 and System V modes are set and 4.2 BSD mode will have precedence. COMPAT_SVID can be set in two ways, by calling setcompat(2) and by compiling the program with the -ZS flag option (see cc(1).

All aspects of 4.2 signals are inherited across fork system calls. 4.2 job control group membership is inherited across exec system calls. When exec is invoked, the inherited 4.2 signals are lost and the signal-handling mechanism returns to System V style. See setcompat(2) for more information.

ERRORS

[EINVAL]	The process has already arranged to catch signals. Normally set42sig is called prior to any other signal activity.

SEE ALSO

cc(1), setcompat(2), sigvec(2), signal(3), termio(7).

NAME

`setbuf`, `setvbuf` — assign buffering to a stream

SYNOPSIS

```
#include <stdio.h>

void setbuf(stream, buf)
FILE *stream;
char *buf;

int setvbuf(stream, buf, type, size)
FILE *stream;
char *buf;
int type, size;
```

DESCRIPTION

`setbuf` may be used after a stream has been opened but before it is read or written. It causes the array pointed to by *buf* to be used instead of an automatically allocated buffer. If *buf* is the NULL pointer input/output will be completely unbuffered.

A constant `BUFSIZ`, defined in the `<stdio.h>` header file, tells how big an array is needed:

```
char buf[BUFSIZ];
```

`setvbuf` may be used after a stream has been opened but before it is read or written. *type* determines how *stream* will be buffered. Legal values for *type* (defined in `stdio.h`) are:

`_IOFBF`	causes input/output to be fully buffered.
`_IOLBF`	causes output to be line buffered; the buffer will be flushed when a newline is written, the buffer is full, or input is requested.
`_IONBF`	causes input/output to be completely unbuffered.

If *buf* is not the NULL pointer, the array it points to will be used for buffering, instead of an automatically allocated buffer. *size* specifies the size of the buffer to be used. The constant `BUFSIZ` in `<stdio.h>` is suggested as a good buffer size. If input/output is unbuffered, *buf* and *size* are ignored.

By default, output to a terminal is line buffered and all other input/output is fully buffered.

RETURN VALUE

If an illegal value for *type* or *size* is provided, `setvbuf` returns a nonzero value. Otherwise, the value returned will be zero.

SEE ALSO

`fopen`(3S), `getc`(3S), `intro`(3), `malloc`(3C), `putc`(3S).

NOTES

A common source of error is allocating buffer space as an "automatic" variable in a code block, and then failing to close the stream in the same block.

`setbuf` allows assignment of a new I/O buffer after the stream has been read (written), and if unflushed data remains in the original buffer. This could lead to a loss of data error.

NAME

setjmp, longjmp — non-local goto

SYNOPSIS

```
#include <setjmp.h>

int setjmp(env)
jmp_buf env;

void longjmp(env, val)
jmp_buf env;
int val;
```

DESCRIPTION

These functions are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.

`setjmp` saves its stack environment in *env* for later use by `longjmp`. The environment type `jmp_buf` is defined in the `<setjmp.h>` header file.

RETURN VALUE

When `setjmp` has been called by the calling process, returns 0.

`longjmp` restores the environment saved by the last call of `setjmp` with the corresponding *env* argument. After `longjmp` is completed, program execution continues as if the corresponding call of `setjmp` (which must not itself have returned in the interim) had just returned the value *val*. `longjmp` cannot cause `setjmp` to return the value 0. If `longjmp` is invoked with a second argument of 0, `setjmp` will return 1. All accessible data have values as of the time `longjmp` was called.

SEE ALSO

`signal`(3).

WARNINGS

`longjmp` fails if it is called when *env* was never primed by a call to `setjmp` or when the last such call is in a function which has since returned.

NAME

setposix — set POSIX compatibility flags

SYNOPSIS

```
int setposix()
```

DESCRIPTION

setposix is equivalent to the following code fragment:

```
#include <compat.h>
setcompat(COMPAT_POSIX);
```

COMPAT_POSIX is equivalent to all of the following:

```
COMPAT_BSDGROUPS
COMPAT_BSDCHOWN
COMPAT_BSDSIGNALS
COMPAT_BSDTTY
COMPAT_SYSCALLS
COMPAT_POSIXPATHTRUNC
COMPAT_EXEC
```

Any non-POSIX compatibility flags that were set prior to the call to setposix are reset.

RETURN VALUE

Upon successful completion, setposix returns the previous compatibility mask. Otherwise, a value of −1 is returned and errno is set to indicate the error.

ERRORS

setposix will return the following error code:

[EINVAL] setposix results in a change in the state of the COMPAT_BSDSIGNALS bit and a signal is currently pending, caught, or held.

SEE ALSO

setcompat(2).

NAME

`setuid`, `setgid` — set user and group IDs

SYNOPSIS

int setuid(*uid*)
int *uid;*

int setgid(*gid*)
int *gid;*

DESCRIPTION

`setuid` (`setgid`) is used to set the real user (group) ID and effective user (group) ID of the calling process.

If the effective user ID of the calling process is superuser, the real user (group) ID and effective user (group) ID are set to *uid* (*gid*).

If the effective user ID of the calling process is not superuser, but its real user (group) ID is equal to *uid* (*gid*), the effective user (group) ID is set to *uid* (*gid*).

If the effective user ID of the calling process is not superuser, but the saved set-user (group) ID from `exec`(2) is equal to *uid* (*gid*), the effective user (group) ID is set to uid(*gid*).

RETURN VALUE

Upon successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and `errno` is set to indicate the error.

ERRORS

`setuid` (`setgid`) will fail if one of the following is true:

[EPERM]	the real user (group) ID of the calling process is not equal to *uid* (*gid*) and its effective user ID is not superuser.
[EINVAL]	The *uid* (*gid*) is out of range.

SEE ALSO

`getuid`(2), `intro`(2), `setregid`(2), `setreuid`(2).

NAME

`sigaction` — examine or change signal action

SYNOPSIS

```
#include &lt;signal.h&gt;

int sigaction(sig, act, oact)
int sig;
struct sigaction *act, *oact;
```

DESCRIPTION

The system defines a set of signals that may be delivered to a process. Signal delivery resembles the occurrence of a hardware interrupt: the signal is blocked from further occurrence, the current process context is saved, and a new one is built. A process may specify a *handler* to which a signal is delivered, or specify that a signal is to be *blocked* or *ignored.* A process may also specify that a default action is to be taken by the system when a signal occurs. Normally, signal handlers execute on the current stack of the process. This may be changed, on a per-handler basis, so that signals are taken on a special "signal stack."

All signals have the same priority. Signal routines execute with the signal that caused their invocation but other signals may yet occur. A global "signal mask" defines the set of signals currently blocked from delivery to a process. The signal mask for a process is initialized from that of its parent (normally 0). It may be changed with a `sigprocmask`(3P) call, or when a signal is delivered to the process.

When a signal condition arises for a process, the signal is added to a set of signals pending for the process. If the signal is not currently *blocked* by the process then it is delivered to the process. When a signal is delivered, the current state of the process is saved, a new signal mask is calculated (as described below), and the signal handler is invoked. The call to the handler is arranged so that if the signal handling routine returns normally, the process resumes execution in the context from before the signal's delivery. If the process wishes to resume in a different context, then it must arrange to restore the previous context itself (see `sigsetjmp`(3P).

`sigaction` allows the calling process to examine or specify the action to be taken on delivery of a signal. *sig* specifies the signal number.

The `sigaction` structure is defined in `<signal.h>`:

```
struct sigaction {
        void (*sa_handler)();
        sigset_t sa_mask;
        int sa_flags;
};
```

If *act* is not NULL, it points to a structure specifying the action to be taken when the signal is delivered. If *oact* is not NULL, the action previously associated with the signal is stored in the location pointed to by *oact*. If *act* is NULL, signal handling is unchanged. When *act* is NULL, `sigaction` can be used to inquire about the current handling of a given signal.

The `sa_flags` field of *act* can be used to modify the delivery of a specific signal. If *sig* is SIGCHLD and the SA_NOCLDSTOP flag is not set in `sa_flags`, a SIGCHLD signal is generated for the calling process if any of its child processes stop. If *sig* is SIGCHLD and the SA_NOCLDSTOP flag is set in `sa_flags`, a SIGCHLD signal is not generated for stopped child processes. If the SA_ONSTACK bit is set in `sa_flags`, the system delivers the signal to the process on a signal stack specified by `sigstack`(2). If the SA_INTERRUPT bit is set in `sa_flags`, system calls interrupted by a signal are not restarted.

When a signal is caught by a signal-catching function, a new signal mask is calculated and installed for the duration of the signal-catching function or until `sigprocmask()` or `sigsuspend()` is called. This mask is formed by taking the union of the current signal mask and the set associated with the action for the signal being delivered, such as `sa_mask`, and then including the signal being delivered. If and when the user's signal handler returns normally, the original signal mask is restored.

Once an action is installed for a specific signal, it remains installed until another action is explicitly requested by another call to `sigaction` or until one of the `exec` functions is called.

`SIGKILL` and `SIGSTOP` cannot be caught or ignored. The set of signals specified by `sa_mask` is not allowed to block these signals. This is silently enforced.

If sigaction fails, no new signal handler is installed.

A/UX POSIX defines the following signals:

SIGHUP	1	hangup
SIGINT	2	interrupt
SIGQUIT	3*	quit
SIGILL	4*	illegal instruction
SIGABRT	6*	aborted
SIGFPE	8*	floating-point exception
SIGKILL	9	kill (cannot be caught, blocked, or ignored)
SIGSEGV	11*	segmentation violation
SIGPIPE	13	write on a pipe with no one to read it
SIGALRM	14	alarm clock
SIGTERM	15	software termination signal
SIGUSR1	16	user defined signal 1
SIGUSR2	17	user defined signal 2
SIGCLD	18•	child status has changed
SIGTSTP	20†	stop signal generated from keyboard
SIGTTIN	21†	background read attempted from control terminal
SIGTTOU	22†	background write attempted to control terminal
SIGSTOP	23†	stop (cannot be caught, blocked, or ignored)
SIGXCPU	24	cpu time limit exceeded
SIGXFSZ	25	file size limit exceeded
SIGCONT	29•	continue after stop (cannot be blocked)

The following signals are also defined:

SIGTRAP	5*	trace trap
SIGIOT	6*	abort
SIGEMT	7*	EMT instruction
SIGBUS	10*	bus error
SIGSYS	12*	bad argument to system call
SIGPWR	19	power-fail restart
SIGVTALRM	26	virtual time alarm (see setitimer(2))
SIGPROF	27	profiling timer alarm (see setitimer(2))
SIGWINCH	28•	window size change
SIGURG	30•	urgent condition present on socket
SIGIO	31•	I/O possible on a descriptor (see fcntl(2))

The starred signals (*) in the list above cause a core image if not caught or ignored.

The default action for a signal may be reinstated by setting sv_handler to SIG_DFL; this default is termination (with a core image for starred signals) except for signals marked with • or

†. Signals marked with • are discarded if the action is SIG_DFL; signals marked with † cause the process to stop. If sv_handler is SIG_IGN, the signal is subsequently ignored, and pending instances of the signal are discarded.

If a caught signal occurs during certain system calls, the call is normally restarted. The affected system calls are read(2) or write(2) on a slow device such as a terminal, but not a file. This behavior may be inhibited by setting the SA_INTERRUPT bit in sa_flags.

After a fork(2), the child inherits all signals, the signal mask, and the signal stack.

execve(2) resets all caught signals to default action and resets all signals to be caught on the user stack. Ignored signals remain ignored; the signal mask remains the same.

RETURN VALUE

On successful completion, a value of 0 is returned. Otherwise, a value of −1 is returned and errno is set to indicate the error.

ERRORS

If any of the following conditions occur, sigaction returns −1 and sets errno to the corresponding value:

[EINVAL]	The value of *sig* is not a valid signal number, or an attempt was made to supply an action for a signal that cannot be caught or ignored.
[EFAULT]	*act* or *oact* is an invalid address. Or both are invalid addresses.

SEE ALSO

exec(2), kill(2), sigsetops(3P), sigprocmask(2P), sigsuspend(3P), sigvec(2).

NAME

`sign, isign, dsign` — Fortran transfer-of-sign intrinsic function

SYNOPSIS

```
integer i, j, k
real r1, r2, r3
double precision dp1, dp2, dp3

k=isign(i, j)
k=sign(i, j)

r3=sign(r1, r2)

dp3=dsign(dp1, dp2)
dp3=sign(dp1, dp2)
```

DESCRIPTION

`isign` returns the magnitude of its first argument with the sign of its second argument. `sign` and `dsign` are its real and double-precision counterparts, respectively. The generic version is `sign`, which devolves to the appropriate type depending on its arguments.

NAME

signal — specify what to do upon receipt of a signal

SYNOPSIS

```
#include <signal.h>

int (*signal(sig, func))()
int sig;
void (*func)();
```

DESCRIPTION

`signal` allows the calling process to choose one of three ways in which it is possible to handle the receipt of a specific signal. *sig* specifies the signal and *func* specifies the choice.

sig can be assigned any one of the following except `SIGKILL`:

SIGHUP	1	hangup
SIGINT	2	interrupt
SIGQUIT	3*	quit
SIGILL	4*	illegal instruction
SIGTRAP	5*	trace trap
SIGIOT	6*	IOT instruction
SIGEMT	7*	EMT instruction
SIGFPE	8*	floating point exception
SIGKILL	9	kill (cannot be caught, blocked, or ignored)
SIGBUS	10*	bus error
SIGSEGV	11*	segmentation violation
SIGSYS	12*	bad argument to system call
SIGPIPE	13	write on a pipe with no one to read it
SIGALRM	14	alarm clock
SIGTERM	15	software termination signal
SIGUSR1	16	user defined signal 1
SIGUSR2	17	user defined signal 2
SIGCLD	18•	child status has changed
SIGPWR	19	power-fail restart
SIGTSTP	20†	stop signal generated from keyboard
SIGTTIN	21†	background read attempted from control terminal
SIGTTOU	22†	background write attempted to control terminal
SIGSTOP	23†	stop (cannot be caught, blocked, or ignored)
SIGXCPU	24	cpu time limit exceeded
SIGXFSZ	25	file size limit exceeded
SIGVTALRM	26	virtual time alarm (see `setitimer`(2))
SIGPROF	27	profiling timer alarm (see `setitimer`(2))
SIGWINCH	28•	window size change

`SIGCONT`	29•	continue after stop (cannot be blocked)
`SIGURG`	30•	urgent condition present on socket
`SIGIO`	31•	I/O is possible on a descriptor (see `fcntl`(2))

The starred signals in the above list cause a core image if not caught or ignored (see below).

Signals marked with • are discarded if the action is `SIG_DFL;` signals marked with † cause the process to stop if the process is part of 4.2 job control.

func is assigned one of three values: `SIG_DFL`, `SIG_IGN`, or a *function-address*. The actions prescribed by these values are as follows:

`SIG_DFL` – terminate process upon receipt of a signal

Upon receipt of the signal *sig*, the receiving process is to be terminated with the following consequences:

All of the receiving process's open file descriptors will be closed.

If the parent process of the receiving process is executing a `wait`, it will be notified of the termination of the receiving process and the terminating signal's number will be made available to the parent process; see `wait`(2).

If the parent process of the receiving process is not executing a `wait`, the receiving process will be transformed into a zombie process (see `exit`(2) for definition of zombie process).

The parent process ID of each of the receiving process's existing child processes and zombie processes will be set to 1. This means the initialization process (see `intro`(2)) inherits each of these processes.

Each attached shared memory segment is detached and the value of `shm_nattach` in the data structure associated with its shared memory identifier is decremented by 1.

For each semaphore for which the receiving process has set a `semadj` value (see `semop`(2)), that `semadj` value is added to the `semval` of the specified semaphore.

If the process has a process, text, or data lock, an unlock is performed (see plock(2)).

An accounting record will be written on the accounting file if the system's accounting routine is enabled; see acct(2).

If the receiving process's process ID, tty group ID, and process group ID are equal, the signal SIGHUP will be sent to all of the processes that have a process group ID equal to the process group ID of the receiving process.

A "core image" will be made in the current working directory of the receiving process if *sig* is one for which an asterisk appears in the above list *and* the following conditions are met:

The effective user ID and the real user ID of the receiving process are equal.

An ordinary file named core exists and is writable or can be created. If the file must be created, it will have the following properties:

a mode of 0666 modified by the file creation mask (see umask(2))

a file owner ID that is the same as the effective user ID of the receiving process

a file group ID that is the same as the effective group ID of the receiving process

SIG_IGN – ignore signal

The signal *sig* is to be ignored.

Note: The signal SIGKILL cannot be ignored.

function-address – catch signal

Upon receipt of the signal *sig*, the receiving process is to execute the signal-catching function pointed to by *func*. The signal number *sig* will be passed as the only argument to the signal-catching function. Additional arguments are passed to the signal-catching function for hardware-generated signals. Before entering the signal-catching function, the value of *func* for the caught signal will be set to SIG_DFL unless the signal is SIGILL, SIGTRAP, or SIGPWR.

Upon return from the signal-catching function, the receiving process will resume execution at the point it was interrupted.

When a signal that is to be caught occurs during a `read`, a `write`, an `open`, or an `ioctl` system call on a slow device (like a terminal; but not a file), during a `pause` system call, or during a `wait` system call that does not return immediately due to the existence of a previously stopped or zombie process, the signal-catching function will be executed and then the interrupted system call may return a −1 to the calling process with `errno` set to `EINTR`. This behavior is the default for 5.2 systems and it may be modified by the `setcompat`(2) system call.

Note: The signal `SIGKILL` cannot be caught.

A call to `signal` cancels a pending signal *sig* except for a pending `SIGKILL` signal.

RETURN VALUE

Upon successful completion, `signal` returns the previous value of *func* for the specified signal *sig*. Otherwise, a value of −1 is returned and `errno` is set to indicate the error.

ERRORS

`signal` will fail if:

`[EINVAL]`	*sig* is an illegal signal number, including `SIGKILL`.

WARNINGS

Two other signals that behave differently than the signals described above exist in this release of the system; they are:

`SIGCLD`	18	death of a child (reset when caught)
`SIGPWR`	19	power fail (not reset when caught)

There is no guarantee that, in future releases of the UNIX system, these signals will continue to behave as described below; they are included only for compatibility with other versions of the UNIX system. Their use in new programs is strongly discouraged.

For these signals, *func* is assigned one of three values: `SIG_DFL`, `SIG_IGN`, or a *function-address*. The actions prescribed by these values of are as follows:

SIG_DFL - ignore signal
The signal is to be ignored.

SIG_IGN - ignore signal
The signal is to be ignored. Also, if *sig* is SIGCLD, the calling process's child processes will not create zombie processes when they terminate; see exit(2).

function-address - catch signal
If the signal is SIGPWR, the action to be taken is the same as that described above for *func* equal to *function-address*. The same is true if the signal is SIGCLD except, that while the process is executing the signal-catching function, any received SIGCLD signals will be queued and the signal-catching function will be continually reentered until the queue is empty.

The SIGCLD affects two other system calls (wait(2), and exit(2)) in the following ways:

wait If the *func* value of SIGCLD is set to SIG_IGN and a wait is executed, the wait will block until all of the calling process's child processes terminate; it will then return a value of −1 with errno set to ECHILD.

exit If in the exiting process's parent process the *func* value of SIGCLD is set to SIG_IGN, the exiting process will not create a zombie process.

When processing a pipeline, the shell makes the last process in the pipeline the parent of the proceeding processes. A process that may be piped into in this manner (and thus become the parent of other processes) should take care not to set SIGCLD to be caught.

SEE ALSO
kill(1), kill(2), pause(2), ptrace(2), setcompat(2), sigvec(2), wait(2), set42sig(3), setjmp(3C).

BUGS
If a repeated signal arrives before the last one can be reset, there is no chance to catch it. However, see the setcompat flag COMPAT_BSDSIGNALS.

The type specification of the routine and its *func* argument are problematical.

The symbols `sighnd` and `sigtrap` are globally defined symbols used by `signal` and are reserved words.

NAME

signal — specify Fortran action on receipt of a system signal

SYNOPSIS

```
integer i
external integer intfnc

call signal(i, intfnc)
```

DESCRIPTION

signal allows a process to specify a function to be invoked upon receipt of a specific signal. The first argument specifies a fault or exception; the second argument specifies the function to be invoked.

SEE ALSO

kill(2), signal(3).

NAME

`sigprocmask` — examine and change blocked signals

SYNOPSIS

```
#include <signal.h>

int sigprocmask(how, set, oset)
int how;
sigset_t *set, oset;
```

DESCRIPTION

`sigprocmask` allows the calling process to examine or change its signal mask. If the value of `set` is not NULL, it points to a set of signals to be used to change the currently blocked set.

The value of *how* indicates the manner in which the set is changed. The permitted values for *how* are:

`SIG_BLOCK`	The resulting set will be the union of the current set and the signal set pointed to by *set*.
`SIG_UNBLOCK`	The resulting set will be the intersection of the current set and the complement of the signal set pointed to by *set*.
`SIG_SETMASK`	The resulting set will be the signal set pointed to by *set*.

If *oset* is not NULL, the previous mask is stored at the location pointed to by *set*. If the value of *set* is NULL, the value of *how* is ignored and the process's signal mask is unchanged. When *set* is NULL, `sigprocmask` can be used to enquire about currently blocked signals.

If there are any pending unblocked signals after the call to `sigprocmask`, at least one of those signals will be delivered before `sigprocmask` returns.

`SIGKILL` and `SIGSTOP` cannot be caught or ignored. `SIGCONT` cannot be ignored. It is not possible to block these signals. This is silently enforced.

RETURN VALUE

Upon successful completion, 0 is returned. Otherwise, −1 is returned and errno is set to indicate the error.

ERRORS

If the following condition occurs, sigprocmask will return −1 and set errno to the corresponding value.

[EINVAL] The value of *how* is invalid.

SEE ALSO

sigaction(3P), sigpending(3P), sigsetops(3P), sigsuspend(3P).

NAME

sigsetjmp, siglongjmp — non-local jumps

SYNOPSIS

```
#include <setjmp.h>

int sigsetjmp(env, savemask)
sigjmp_buf env;
int savemask;

void siglongjmp(env, val)
sigjmp_buf env;
int val;
```

DESCRIPTION

These functions are useful for dealing with errors and interrupts encountered in a low-level subroutine of a program.

`sigsetjmp` saves its stack environment in *env* for later use by `siglongjmp`. If the value of *savemask* is not zero, `sigsetjmp` also saves the process's current signal mask as part of the calling environment. The environment type `sigjmp_buf` is defined in the `<setjmp.h>` header file.

`siglongjmp` restores the environment saved by the last call of `sigsetjmp` with the corresponding *env* argument. If *env* was initialized by a call to `sigsetjmp` with a non-zero value for *savemask*, `siglongjmp` also restores the saved signal mask.

RETURN VALUE

When `sigsetjmp` has been invoked by the calling process, zero is returned.

After `siglongjmp` is completed, program execution continues as if the corresponding call of `sigsetjmp` (which must not itself have returned in the interim) had just returned the value *val*. `siglongjmp` cannot cause `sigsetjmp` to return the value zero. If *val* is zero, `sigsetjmp` returns 1. All accessible data have values as of the time `siglongjmp` was called.

WARNINGS

`siglongjmp` fails if *env* was never initialized by a call to `sigsetjmp` or when the last such call is in a function which has since returned.

SEE ALSO

sigaction(3P), sigprocmask(3P), sigsuspend(3P).

NAME

sigaddset, sigdelset, sigismember, sigfillset, siginitset — manipulate signal sets

SYNOPSIS

```
#include &lt;signal.h&gt;

int sigaddset(set, signo)
sigset_t *set;
int signo;

int sigdelset(set, signo)
sigset_t *set;
int signo;

int sigismember(set, signo)
sigset_t *set;
int signo;

int sigfillset(set)
sigset_t *set;

int sigemptyset(set)
sigset_t *set;
```

DESCRIPTION

These routines manipulate sets of signals. They operate on data objects addressable by the application, not on any set of signals known to the system. The signal set modified by these rountines may be used as a parameter to sigagction(3P), sigprocmask(3P), sigpending(3P), or sigsispend(3P). sigaddset adds the signal specified by pointed to by *set*.

sigdelset deletes the signal specified by *signo* from the set pointed to by *set*.

POSIX defines the following signals:

SIGABRT	SIGPIPE	SIGCLD
SIGALRM	SIGQUIT	SIGCONT
SIGFPE	SIGSEGV	SIGSTOP
SIGHUP	SIGTERM	SIGTSTP
SIGILL	SIGUSR1	SIGTTIN
SIGINT	SIGUSR2	SIGTTOU
SIGKILL		

`sigfillset` initializes the signal set pointed to by *set* so that all POSIX-defined signals are included.

`sigemptyset` initializes the signal set pointed to by *set* so that all the POSIX-defined signals are excluded. Applications must call `sigemptyset` for each object of type `sigset_t` before any other use of the object.

`sigismember` tests whether the signal specified by *signo* is a member of the set pointed to by *set*.

RETURN VALUE

On successful completion, `sigismember` returns 1 if the specified signal is a member of the specified set and returns 0 if it is not. On successful completion, each of the other functions returns 0. For all the functions listed, if an error is detected, `sigaddset`, `sigdelset`, and `sigismember` returns −1 and set `errno` to indicate the error.

ERRORS

If any of the following conditions occur, the function returns −1 and sets `errno` to the corresponding value:

`[EINVAL]`	The value of *signo* is not a valid signal number.
`[EFAULT]`	*set* is an invalid address.

SEE ALSO

`sigaction`(3P), `sigpending`(3P), `sigprocmask`(2P), `sigsuspend`(3P).

NAME

`sigsuspend` — wait for a signal

SYNOPSIS

```
#include <signal.h>

int sigsuspend(sigmask)
sigset_t *sigmask;
```

DESCRIPTION

`sigsuspend` replaces the process's signal mask with the set of signals pointed to by `sigmask` and then suspends the process until delivery of a signal whose action is either to execute a signal-catching function or to terminate the process.

If the action is to terminate the process, `sigsuspend` will not return. If the action is to execute a signal-catching function, `sigsuspend` will return after the signal-catching function returns, with the signal mask restored to the set that existed prior to the `sigsuspend` call.

`SIGKILL` and `SIGSTOP` cannot be caught or ignored. `SIGCONT` cannot be ignored. It is not possible to block these signals. This is silently enforced.

RETURN VALUE

Since `sigsuspend` suspends process execution indefinitely, there is no successful completion return value. If `sigsuspend` returns, it will return −1 and `errno` will be set to indicate the error.

ERRORS

If the following condition occurs, `sigsuspend` will return −1 and set `errno` to the corresponding value.

[EINTR]	A signal is caught by the calling process and control is returned from the signal-catching function.

SEE ALSO

`pause`(2), `sigaction`(3P), `sigpending`(3P), `sigprocmask`(2P), `sigsetops`(3P).

NAME

`sin, dsin, csin` — Fortran sine intrinsic function

SYNOPSIS

```
real r1, r2
double precision dp1, dp2
complex cx1, cx2

r2=sin(r1)

dp2=dsin(dp1)
dp2=sin(dp1)

cx2=csin(cx1)
cx2=sin(cx1)
```

DESCRIPTION

`sin` returns the real sine of its real argument. `dsin` returns the double-precision sine of its double-precision argument. `csin` returns the complex sine of its complex argument. The generic `sin` function becomes `dsin` or `csin` as required by argument type.

SEE ALSO

`trig`(3M).

NAME

`sinh`, `dsinh` — Fortran hyperbolic sine intrinsic function

SYNOPSIS

```
real r1, r2
double precision dp1, dp2

r2=sinh(r1)

dp2=dsinh(dp1)
dp2=sinh(dp1)
```

DESCRIPTION

`sinh` returns the real hyperbolic sine of its real argument. `dsinh` returns the double-precision hyperbolic sine of its double-precision argument. The generic form `sinh` may be used to return a double-precision value given a double-precision argument.

SEE ALSO

`sinh`(3M).

NAME

sinh, cosh, tanh — hyperbolic functions

SYNOPSIS

```
#include <math.h>

double sinh(x)
double x;

double cosh(x)
double x;

double tanh(x)
double x;
```

DESCRIPTION

sinh, cosh, and tanh return, respectively, the hyperbolic sine, cosine, and tangent of their argument.

RETURN VALUE

sinh and cosh return HUGE (and sinh may return -HUGE for negative *x*) when the correct value would overflow and set errno to ERANGE.

These error-handling procedures may be changed with the function matherr(3M).

SEE ALSO

matherr(3M).

NAME

sleep — suspend execution for interval

SYNOPSIS

```
unsigned sleep(seconds)
unsigned seconds;
```

DESCRIPTION

sleep suspends the current process from execution for the number of *seconds* specified by the argument. The actual suspension time may be less than that requested for two reasons: (1) scheduled wakeups occur at fixed 1-second intervals, (on the second, according to an internal clock) and (2) any caught signal will terminate sleep following execution of the signal catching routine. The suspension time may be longer than requested by an arbitrary amount, due to the scheduling of other activity in the system. The value returned by sleep is the "unslept" amount (the requested time minus the time actually slept) in case the caller had an alarm set to go off earlier than the end of the requested sleep time or in case there is premature arousal due to another caught signal.

The routine is implemented by setting an alarm signal and pausing until it (or some other signal) occurs. The previous state of the alarm signal is saved and restored. The calling program may have set up an alarm signal before calling sleep. If the sleep time exceeds the time before the alarm signal, the process sleeps only until the alarm signal would have occurred and the caller's alarm catch routine is executed just before the sleep routine returns. If the sleep time is less than the time before the calling program's alarm, the prior alarm time is reset to go off at the same time it would have without the intervening sleep.

SEE ALSO

alarm(2), pause(2), signal(3).

NAME

slots — ROM library functions

SYNOPSIS

cc [*flags*] *files* -lslots [*libraries*]

DESCRIPTION

The routines in the slots library provide access to board slot ROM from either user or kernel processes. Calls to library routines do not require knowledge of either the board ROM configuration or the ROM addressing requirements.

USER FUNCTIONS

slot_PRAM_init(*slot, data*)
: Read the PRAM init structure for *slot* into the buffer pointed to by *data*.

slot_board_flags(*slot*)
: Read and return the board flags for *slot*.

slot_board_id(*slot*)
: Read and return the board ID number for *slot*.

slot_board_name(*slot, data, size*)
: Read up to *size* bytes of the board name string for *slot* into the buffer pointed to by *data*.

slot_board_type(*slot, data*)
: Read and return the unsigned 64 bit or 8 byte board type for *slot* into the buffer pointed to by *data*.

slot_ether_addr(*slot, data*)
: For *slot* read 6 bytes of ethernet address into the buffer pointed to by *data*.

slot_primary_init(*slot, data*)
: For *slot* read the primary init structure into the buffer pointed to by *data*.

slot_part_num(*slot, data, size*)
: For *slot* get *size* bytes of the part number string into the buffer pointed to by *data*.

slot_rev_level(*slot, data, size*)
: For *slot* get *size* bytes of the revision level of the ROM into the buffer pointed to by *data*.

slot_serial_number(*slot, data, size*)
: For *slot* get *size* bytes of serial number string into the buffer

pointed to by *data*.

`slot_vendor_id(`*slot*`,` *data*`,` *size*`)`
For *slot* read *size* bytes of vendor ID string into the buffer pointed to by *data*.

UTILITY FUNCTIONS

`slot_board_vendor_info(`*kind*`,` *slot*`,` *data*`,` *size*`)`
For *slot* get *size* bytes of the vendor information string of type *kind* into the buffer pointed to by *data*.

`slot_byte(`*address*`)`
Return the byte located at *address*.

`slot_data(`*slot*`,` *kind*`,` *request*, *data*`, size)`
For *slot*lot, read *size* BITS of data for resource of type *kind* from the resource list item of type *request* and put it into the location pointed to by *data*.

`slot_directory(`*slot*`,` *data*`,` *size*`)`
For *slot* read the resource directory into the buffer of *size* entries pointed to by *data*.

`slot_long(`*address*`,` *data*`)`
Return 32 bits of data from *address* offset by *data*.

`slot_resource(`*address*`,` *kind*`, request,` *data*`,` *size*`)`
For ROM starting at base *address* read *size* bytes of the *request* resource item from the *kind* resource into the buffer pointed to by *data*.

`slot_resource_list(`*address*`,` *kind*`,` *data*`,` *size*`)`
For ROM starting at base *address* read *size* entries of resource list of *kind* into the buffer pointed to by *data*.

`slot_structure(`*address*`,` *from*`,` *data*`,` *size*`)`
From ROM starting at *address* plus the offset in parameter *from* read *size* bytes of data into the buffer pointed to by *data*.

`slot_word(`*address*`)`
Return 16 bits of data located at *address*.

LOW LEVEL FUNCTIONS

`slot_seg_violation()`
This routine is passed to `slot_catch` to handle bus errors.

`slot_catch(`*kind*`,` *routine*`)`
Setup *routine* to handle interrupts of type *kind*.

`slot_ignore` (*kind*)
Return the system to default handling of interrupts of type *kind*.

`slot_address` (*slot*)
Returns a computed ROM base address for *slot*.

`slot_bytelane` (*address*, *bytelane*)
Return the ROM bytelane byte into *bytelane* for ROM starting at *address*.

`slot_calc_pointer` (*current*, *offset*)
Return a ROM pointer *offset* bytes from *current*.

`slot_rom_data` (*address*, *width*, *data*)
Starting with *address* fill the buffer pointed to by *data* with *width* bytes of data.

`slot_check_crc` (*top*, *fhp*, *bytelane*)
Check the CRC for the ROM with base address *top* using the format header information pointed to by *fhp* and the byte lane information in *bytelane*.

`slot_header` (*address*, *format_hdrp*)
Read the ROM format header into the buffer pointed to by *format_hdrp* for the ROM starting at base address *address*.

SEE ALSO

Building A/UX Device Drivers

NOTE

The slots library is only accessible to processes with superuser privileges due to the required `phys` call to access board ROM.

NAME

spray — scatter data in order to check the network

SYNOPSIS

```
#include <rpcsvc/spray.h>
```

DESCRIPTION

RPC INFO

Program number: SPRAYPROG

xdr routines:

```
xdr_sprayarr(xdrs, arr);
    XDR *xdrs;
    struct sprayarr *arr;
xdr_spraycumul(xdrs, cumul);
    XDR *xdrs;
    struct spraycumul *cumul;
```

Procs:

SPRAYPROC_SPRAY
: Takes no arguments; returns no value. Increments a counter in server daemon. The server does not return this call, so the caller should have a timeout of 0.

SPRAYPROC_GET
: Takes no arguments; returns structure spraycumul with value of counter and clock.

SPRAYPROC_CLEAR
: Takes no arguments and returns no value. Zeros out counter and clock.

Versions:
SPRAYVERS_ORIG

Structures:

```
struct spraycumul {
    unsigned counter;
    struct timeval clock;
};
struct sprayarr {
    int *data,
    int lnth
};
```

SEE ALSO

spray(1M), sprayd(1M).

NAME

sputl, sgetl — access long integer data in a machine independent fashion

SYNOPSIS

```
void sputl(value, buffer)
long value;
char *buffer;

long sgetl(buffer)
char *buffer;
```

DESCRIPTION

sputl takes the 4 bytes of the long integer *value* and places them in memory, starting at the address pointed to by *buffer*. The ordering of the bytes is the same across all machines.

sgetl retrieves the 4 bytes in memory, starting at the address pointed to by *buffer*, and returns the long integer value in the byte ordering of the host machine.

Use of sputl and sgetl provide a machine independent way of storing long numeric data in a file in binary form without conversion to characters.

A program that uses these functions must be loaded with the object file access routine library libld.a.

SEE ALSO

ar(4).

NAME

sqrt, dsqrt, csqrt — Fortran square root intrinsic function

SYNOPSIS

```
real r1, r2
double precision dp1, dp2
complex cx1, cx2

r2=sqrt(r1)

dp2=dsqrt(dp1)
dp2=sqrt(dp1)

cx2=csqrt(cx1)
cx2=sqrt(cx1)
```

DESCRIPTION

`sqrt` returns the real square root of its real argument. `dsqrt` returns the double-precision square root of its double-precision argument. `csqrt` returns the complex square root of its complex argument. `sqrt`, the generic form, will become `dsqrt` or `csqrt` as required by its argument type.

SEE ALSO

exp(3M).

NAME

`ssignal`, `gsignal` — software signals

SYNOPSIS

```
#include <signal.h>

int (*ssignal(sig, action))()
int sig, (*action)();

int gsignal(sig)
int sig;
```

DESCRIPTION

`ssignal` and `gsignal` implement a software facility similar to `signal`(3). This facility is used by the Standard C Library to enable users to indicate the disposition of error conditions; it is also made available to users for their own purposes.

Software signals made available to users are associated with integers in the inclusive range 1 through 15. A call to `ssignal` associates a procedure, *action*, with the software signal, *sig*; the software signal, *sig*, is raised by a call to `gsignal`. Raising a software signal causes the action established for that signal to be taken.

The first argument to `ssignal` is a number identifying the type of signal for which an action is to be established. The second argument defines the action; it is either the name of a user-defined *action* function or one of the manifest constants `SIG_DFL` (default) or `SIG_IGN` (ignore). `ssignal` returns the action previously established for that signal type; if no *action* has been established or the signal number (*sig*) is illegal, `ssignal` returns `SIG_DFL`.

`gsignal` raises the signal identified by its argument, *sig*:

> If an *action* function has been established for *sig*, then that *action* is reset to `SIG_DFL` and the *action* function is entered with argument *sig*. `gsignal` returns the value returned to it by the *action* function.
>
> If the *action* for *sig* is `SIG_IGN`, `gsignal` returns the value 1 and takes no other action.
>
> If the *action* for *sig* is `SIG_DFL`, `gsignal` returns the value 0 and takes no other action.

If *sig* has an illegal value or no *action* was ever specified for *sig*, `gsignal` returns the value 0 and takes no other action.

SEE ALSO

`sigvec`(2), `signal`(3).

NOTES

There are some additional signals with numbers outside the range 1 through 15 which are used by the Standard C Library to indicate error conditions. Thus, some signal numbers outside the range 1 through 15 are legal, although their use may interfere with the operation of the Standard C Library.

NAME

strcat, strncat, strcmp, strncmp, strcpy, strncpy, strlen, strchr, strrchr, strpbrk, strspn, strcspn, strtok — string operations

SYNOPSIS

```
#include <string.h>

char *strcat(s1, s2)
char *s1, *s2;

char *strncat(s1, s2, n)
char *s1, *s2;
int n;

int strcmp(s1, s2)
char *s1, *s2;

int strncmp(s1, s2, n)
char *s1, *s2;
int n;

char *strcpy(s1, s2)
char *s1, *s2;

char *strncpy(s1, s2, n)
char *s1, *s2;
int n;

int strlen(s)
char *s;

char *strchr(s, c)
char *s;
int c;

char *strrchr(s, c)
char *s;
int c;

char *strpbrk(s1, s2)
char *s1, *s2;

int strspn(s1, s2)
char *s1, *s2;

int strcspn(s1, s2)
char *s1, *s2;
```

```
char *strtok(s1, s2)
char *s1, *s2;
```

DESCRIPTION

The arguments *s1*, *s2*, and *s* point to strings (arrays of characters terminated by a null character). The functions `strcat`, `strncat`, `strcpy`, and `strncpy` all alter *s1*. These functions do not check for overflow of the array pointed to by *s1*.

`strcat` appends a copy of string *s2* to the end of string *s1*. `strncat` appends at most *n* characters. Each function returns a pointer to the null-terminated result.

`strcmp` performs a lexicographical comparison of its arguments and returns an integer less than, equal to, or greater than 0, when *s1* is less than, equal to, or greater than *s2*, respectively. `strncmp` makes the same comparison but looks at a maximum of *n* characters.

`strcpy` copies string *s2* to string *s1*, stopping after the null character has been copied. `strncpy` copies exactly *n* characters, truncating *s2* or adding null characters to *s1* if necessary. The result is not null-terminated if the length of *s2* is *n* or more. Each function returns *s1*.

`strlen` returns the number of characters in *s*, not including the terminating null character.

`strchr` (`strrchr`) returns a pointer to the first (last) occurrence of character *c* in string *s*, or a NULL pointer if *c* does not occur in the string. The null character terminating a string is considered to be part of the string.

`strpbrk` returns a pointer to the first occurrence in string *s1* of any character from string *s2*, or a NULL pointer if no character from *s2* exists in *s1*.

`strspn` (`strcspn`) returns the length of the initial segment of string *s1* which consists entirely of characters from (not from) string *s2*.

`strtok` considers the string *s1* to consist of a sequence of zero or more text tokens separated by spans of one or more characters from the separator string *s2*. The first call (with pointer *s1* specified) returns a pointer to the first character of the first token, and writes a null character into *s1* immediately following the returned token. The function keeps track of its position in the string between separate calls, so that on subsequent calls (which must be made with a NULL pointer as the first argument) it works through the string *s1* immediately following that token. This can be continued until no tokens remain. The separator string *s2* may be different from call to call. When no token remains in *s1*, a NULL pointer is returned.

NOTES

For user convenience, some of these functions are declared in the optional `<string.h>` header file.

BUGS

`strcmp` uses native character comparison. Thus the sign of the value returned when one of the characters has its high-order bit set is implementation-dependent.

All string movement is performed character by character starting at the left. Thus overlapping moves toward the left will work as expected, but overlapping moves to the right may yield surprises.

NAME

strtod — convert string to double-precision number

SYNOPSIS

```
double strtod(str, ptr)
char *str, **ptr;
```

DESCRIPTION

strtod returns as a double-precision floating-point number, the value represented by the character string pointed to by *str*. The string is scanned up to the first unrecognized character.

strtod recognizes an optional string of "white-space" characters (as defined by isspace in ctype(3C)), then an optional sign, then a string of digits optionally containing a decimal point, then an optional e or E followed by an optional sign or space, followed by an integer.

If the value of *ptr* is not (char **)NULL, a pointer to the character terminating the scan is returned in the location pointed to by *ptr*. If no number can be formed, **ptr* is set to *str*, and zero is returned.

SEE ALSO

bstring(3), atof(3C), ctype(3C), memcpy(3C), scanf(3S), string(3C). strtol(3C).

DIAGNOSTICS

If the correct value would cause overflow, ±HUGE is returned (according to the sign of the value), and errno is set to ERANGE.
If the correct value would cause underflow, zero is returned and errno is set to ERANGE.

NAME

strtol, atol, atoi — convert string to integer

SYNOPSIS

```
long strtol(str, ptr, base)
char *str, **ptr;
int base;

long atol(str)
char *str;

int atoi(str)
char *str;
```

DESCRIPTION

strtol returns as a long integer the value represented by the character string pointed to by *str*. The string is scanned up to the first character inconsistent with the base. Leading white-space characters (blanks and tabs) are ignored.

If the value of *ptr* is not (char **)NULL, a pointer to the character terminating the scan is returned in the location pointed to by *ptr*. If no integer can be formed, zero is returned.

If *base* is positive (and not greater than 36), it is used as the base for conversion. After an optional leading sign, leading zeros are ignored; a leading 0x or 0X is ignored if *base* is 16.

If *base* is zero, the string itself determines the base. After an optional leading sign, a leading zero indicates octal conversion and a leading 0x or 0X indicates hexadecimal conversion; otherwise, decimal conversion is used.

Truncation from long to int can take place upon assignment or by an explicit cast.

atol(*str*) is equivalent to:

```
strtol(str, (char **)NULL, 10)
```

atoi(*str*) is equivalent to:

```
(int)strtol(str, (char **)NULL, 10)
```

SEE ALSO

ctype(3C), scanf(3S), strtod(3C).

BUGS

Overflow conditions are ignored.

NAME

swab — swap bytes

SYNOPSIS

```
void swab(from, to, nbytes)
char *from, *to;
int nbytes;
```

DESCRIPTION

swab copies *nbytes* bytes referenced by *from* to the array referenced by *to*, exchanging adjacent even and odd bytes. It is useful for carrying binary data between PDP-11s and other machines. *nbytes* should be even and non-negative. If *nbytes* is odd and positive, swab uses *nbytes*−1 instead. If *nbytes* is negative, swab does nothing.

NAME

sysconf — get configurable system variables

SYNOPSIS

```
#include <unistd.h>

long sysconf(name)
int name;
```

DESCRIPTION

sysconf allows an application to determine the current value of a configurable system variable.

name represents the system variable to be queried. Allowable values for *name* are:

```
_SC_ARG_MAX
_SC_CHILD_MAX
_SC_CLK_TCK
_SC_NGROUPS_MAX
_SC_OPEN_MAX
_SC_PASS_MAX
_SC_PID_MAX
_SC_UID_MAX
_SC_EXIT_SIGHUP
_SC_JOB_CONTROL
_SC_KILL_PID_NEG1
_SC_KILL_SAVED
_SC_PGID_CLEAR
_SC_SAVED_IDS
_SC_VERSION
```

RETURN VALUE

sysconf returns the current value of the specified variable. The value returned will not be more restrictive than the value described to the application when it was compiled with <limits.h> or <unistd.h>. The value will not change during the lifetime of the calling process.

ERRORS

If *name* is not defined on the system or *name* is invalid, sysconf will return −1.

SEE ALSO

pathconf(3P).

NAME

system — issue a shell command from Fortran

SYNOPSIS

```
character *N c

call system(c)
```

DESCRIPTION

system causes its character argument to be given to sh(1) as input, as if the string had been typed at a terminal. The current process waits until the shell has completed.

SEE ALSO

sh(1), exec(2), system(3S).

NAME

system — issue a shell command

SYNOPSIS

```
#include &lt;stdio.h&gt;

int system(string)
char *string;
```

DESCRIPTION

system causes *string* to be given to sh(1) input, as if the string had been typed as a command at a terminal. The current process waits until the shell has completed and then returns the exit status of the shell.

RETURN VALUE

system forks to create a child process that in turn performs exec(2) on /bin/sh in order to execute *string*. If fork or exec fails, system returns a negative value and sets errno. If fork and exec succeed, the exit status of the shell is returned.

FILES

/bin/sh

SEE ALSO

sh(1), exec(2).

NAME

`tan`, `dtan` — Fortran tangent intrinsic function

SYNOPSIS

`real` *r1*, *r2*
`double precision` *dp1*, *dp2*

r2=`tan`(*r1*)

dp2=`dtan`(*dp1*)
dp2=`ftan`(*dp1*)

DESCRIPTION

`tan` returns the real tangent of its real argument. `dtan` returns the double-precision tangent of its double-precision argument. The generic `tan` function becomes `dtan` as required with a double-precision argument.

SEE ALSO

`trig`(3M).

NAME

`tanh`, `dtanh` — Fortran hyperbolic tangent intrinsic function

SYNOPSIS

`real` *r1*, *r2*
`double precision` *dp1*, *dp2*

r2=`tanh`(*r1*)

dp2=`dtanh`(*dp1*)
dp2=`tanh`(*dp1*)

DESCRIPTION

`tanh` returns the real hyperbolic tangent of its real argument. `dtanh` returns the double-precision hyperbolic tangent of its double precision argument. The generic form `tanh` may be used to return a double-precision value given a double-precision argument.

SEE ALSO

`sinh`(3M).

NAME

`tcdrain`, `tcflow`, `tcflush`, `tcsendbreak` — line control functions

SYNOPSIS

```
#include <termios.h>

int tcdrain(fildes)
int fildes;

int tcflow(fildes, action)
int fildes, action;

int tcflush(fildes, queue_selector)
int fildes, queue_selector;

int tcsendbreak(fildes, duration)
int fildes, duration;
```

DESCRIPTION

`tcdrain` causes the process to wait until all output written to the object indicated by *fildes* has been transmitted.

`tcflow` will suspend transmission or reception of data on the object indicated by *fildes*, depending on the value of *action*. If *action* is `TCOOFF`, output will be suspended. If *action* is `TCOON`, suspended output will be restarted. If *action* is `TCIOF`, input will be suspended. If *action* is `TCION`, suspended input will be restarted.

`tcflush` will discard data written to the object indicated by *fildes* but not transmitted, or data received but not read, depending on the value of *queue_selector*. If *queue_selector* is `TCIFLUSH` data received, but not read, will be flushed. If *queue_selector* is `TCOFLUSH` data written, but not transmitted, will be flushed. If *queue_selector* is `TCIOFLUSH` both data received, but not read, and data written, but not transmitted, will be flushed.

`tcsendbreak` will assert a break condition on the serial line associated with *fildes* depending on the value of *duration*. If *duration* is zero, the break condition will be asserted for 0.25 seconds. If *duration* is not zero, no break will be sent.

RETURN VALUE

Upon successful completion, zero is returned. Otherwise, −1 is returned and `errno` is set to indicate the error.

ERRORS

If any of the following conditions occur, −1 will be returned and `errno` will be set to the corresponding value.

[EBADF]	*fildes* is not a valid file descriptor.
[EINVAL]	The device does not support the function or if the function called was `tcflush`, *queue_selector* is invalid.
[ENOTTY]	The file associated with *fildes* is not a terminal.

In addition to those listed already, `tcdrain` will report the following error.

[EINTR]	`tcdrain` was interrupted by a signal.

SEE ALSO

`termios`(7P).

NAME

tcgetattr, tcsetattr — get and set the terminal state

SYNOPSIS

```
#include <termios.h>

int tcgetattr(fildes, termios-p)
int fildes;
struct termio *termio-p;

int tcsetattr(fildes, optional-actions, termio-p)
int fildes, optional-actions;
struct termio *termio-p;
```

DESCRIPTION

tcgetattr retrieves the parameters associated with the device indicated by *fildes* and stores them in the termios structure indicated by *termios-p*.

tcsetattr sets the parameters associated with the terminal using the information in the termios structure pointed to by *termios-p*. The action taken is dependent on the value of *optional-actions*. If *optional-actions* is TCSANOW, the change occurs immediately. If *optional-actions* is TCSADRAIN, the change occurs after all output written to *fildes* has been transmitted. TCSADRAIN should be used when changing parameters that affect output. If *optional-actions* is TCSAFLUSH, the change occurs after all output written to the object indicated by *fildes* has been transmitted; all input that has been received but not read is discarded before the change is made.

tcgettattr is allowed from a background process; however, the terminal attributes may be changed later by a foreground process.

RETURN VALUE

On successful completion, a value of 0 is returned. Otherwise, −1 is returned and errno is set to indicate the error.

ERRORS

If any of the following conditions occur, tcgetattr and tcsetattr return −1 and set errno to the corresponding value:

[EBADF] The file descriptor *fildes* is not valid.

[EINVAL] The device does not support the function called, or if the function called was `tcsetattr`, *optional-actions* is an invalid value.

[ENOTTY] The file associated with *fildes* is not a terminal.

SEE ALSO

`cfgetospeed`(3P), `termios`(7P).

NAME

tcgetpgrp — get distinguished process group ID

SYNOPSIS

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

pid_t tcgetpgrp(fildes)
int fildes;
```

DESCRIPTION

`tcgetpgrp` is part of the POSIX Job Control option.

`tcgetpgrp` returns the value of the process group ID of the foreground process group associated with the terminal. `tcgetpgrp` may be called from a process that is a member of a background process group; however, the information may be subsequently changed by a process that is a member of a foreground process group.

RETURN VALUE

On successful completion, `tcgetpgrp` returns the process group ID of the foreground process group associated with the terminal. Otherwise, −1 is returned and `errno` is set to indicate the error.

ERRORS

If any of the following conditions occur, `tcgetpgrp` will return −1 and set `errno` to the corresponding value.

[EBADF]	The file descriptor *fildes* is not valid.
[EINVAL]	`tcgetpgrp` is not permitted for the device associated with *fildes*.
[ENOTTY]	The calling process does not have a controlling terminal, or the file is not the controlling terminal.

SEE ALSO

`setsid`(2P), `setpgid`(2P), `tcsetpgrp`(3P).

NAME

tcsetpgrp — set distinguished process group ID

SYNOPSIS

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

int tcsetpgrp (fildes, pgrp-id)
int fildes;
pid_t pgrp-id;
```

DESCRIPTION

tcsetpgrp is part of the POSIX Job Control Option.

If the process has a controlling terminal, tcsetpgrp sets the distinguished process group ID associated with the terminal to *pgrp-id*. The file associated with *fildes* must be the controlling terminal of the calling process, and the controlling terminal must be currently associated with the session of the calling process. The *pgrp-id* must match a process group ID of a process in the same session as the calling process.

RETURN VALUE

On successful completion, tcsetpgrp returns 0. Otherwise, −1 is returned and errno is set to indicate the error.

ERRORS

[EBADF]	The file descriptor *fildes* is not valid.
[EINVAL]	tcsetpgrp is not permitted for the device associated with *fildes*, or the value of *pgrp-id* is less than or equal to 0 or exceeds PID_MAX.
[ENOTTY]	The calling process does not have a controlling terminal, or the file is not the controlling terminal.
[EPERM]	*pgrp-id* is greater than 0 and less than or equal to PID_MAX, and there is no process in the process group indicated by *pgrp-id* that has the same controlling terminal as the calling process.

SEE ALSO

setsid(2P), setpgid(2P), tcgetpgrp(3P).

NAME

tgetent, tgetnum, tgetflag, tgetstr, tgoto, tputs — terminal independent operation routines

SYNOPSIS

```
char PC;
char *BC;
char *UP;
short ospeed;

int tgetent(bp, name)
char *bp, *name;

int tgetnum(id)
char *id;

int tgetflag(id)
char *id;

char *tgetstr(id, area)
char *id, **area;

char *tgoto(cm, destcol, destline)
char *cm;
int destcol;
int destline;

int tputs(cp, affcnt, outc)
char *cp;
int affcnt;
int(*outc)();
```

DESCRIPTION

These functions extract and use capabilities from the terminal capability database `termcap`(4). Note that these are low-level routines.

`tgetent` extracts the entry for terminal *name* into the buffer at *bp*. *bp* should be a character buffer of size 1024 and must be retained through all subsequent calls to `tgetnum`, `tgetflag`, and `tgetstr`. `tgetent` returns −1 if it cannot open the `termcap` file, 0 if the terminal name given does not have an entry, and 1 if successful. It looks in the environment for a `TERMCAP` variable. If a variable is found whose value does not begin with a slash and the terminal type *name* is the same as the environment string `TERM`, the `TERMCAP` string is used instead of reading the `termcap` file. If the value does begin with a slash, the string is used as a pathname rather than `/etc/termcap`. This can speed up entry into programs that call tgetent. It can also help debug new terminal descriptions or be used to make one for your terminal if you can't write the file /etc/termcap.

tgetnum gets the numeric value of capability *id*, returning −1 if is not given for the terminal. tgetflag returns 1 if the specified capability is present in the terminal's entry, 0 if it is not. tgetstr gets the string value of capability *id*, placing it in the buffer at *area*, advancing the *area* pointer. It decodes the abbreviations for this field described in termcap(4), except for cursor addressing and padding information.

tgoto returns a cursor addressing string decoded from *cm* to go to column *destcol* in line *destline*. It uses the external variables UP (from the up capability) and BC (if bc is given rather than bs) if necessary to avoid placing \n, ^D, or ^@ in the returned string. (Programs that call tgoto should be sure to turn off the XTABS bit(s), since tgoto may now output a tab. Note that programs using termcap should in general turn off XTABS anyway since some terminals use CONTROL-I for other functions, such as nondestructive space.) If a % sequence is given which is not understood, then tgoto returns "OOPS".

tputs decodes the leading padding information of the string *cp*; affcnt gives the number of lines affected by the operation, or 1 if this is not applicable; outc is a routine that is called with each character in turn. The external variable ospeed should contain the output speed of the terminal as encoded by stty (1). The external variable PC should contain a pad character to be used (from the pc capability) if a null (^@) is inappropriate.

FILES

/lib/libtermcap.a
/etc/termcap

SEE ALSO

ex(1), termcap(4).

NAME

tmpfile — create a temporary file

SYNOPSIS

```
#include <stdio.h>

FILE *tmpfile()
```

DESCRIPTION

tmpfile creates a temporary file using a name generated by tmpnam(3S), and returns a corresponding FILE pointer. The file is automatically deleted when the process using it terminates. The file is opened for update ("w+"). tmpfile calls fopen and so returns any error code passed to it from fopen.

RETURN VALUE

If the temporary file cannot be opened, an error message is printed using perror(3C), and a NULL pointer is returned.

SEE ALSO

creat(2), unlink(2), fopen(3S), mktemp(3C), perror(3C), tmpnam(3S).

NAME

`tmpnam`, `tempnam` — create a name for a temporary file

SYNOPSIS

```
#include <stdio.h>

char *tmpnam(s)
char *s;

char *tempnam(dir, pfx)
char *dir, *pfx;
```

DESCRIPTION

These functions generate filenames that can safely be used for a temporary file.

`tmpnam` always generates a filename using the pathname defined as `p_tmpdir` in the `<stdio.h>` header file. If *s* is NULL, *tmpnam* leaves its result in an internal static area and returns a pointer to that area. The next call to `tmpnam` will destroy the contents of the area. If *s* is not NULL, it is assumed to be the address of an array of at least `l_tmpnam` bytes, where `l_tmpnam` is a constant defined in `<stdio.h>`; `tmpnam` places its result in that array and returns *s*.

`tempnam` allows the user to control the choice of a directory. The argument *dir* points to the pathname of the directory in which the file is to be created. If *dir* is NULL or points to a string which is not a pathname for an appropriate directory, the pathname defined as `p_tmpdir` in the `<stdio.h>` header file is used. If that pathname is not accessible, `/tmp` will be used as a last resort. This entire sequence can be upstaged by providing an environment variable `TMPDIR` in the user's environment, whose value is a pathname for the desired temporary-file directory.

Many applications prefer that names of temporary files contain favorite initial letter sequences. Use the *pfx* argument for this. This argument may be NULL or point to a string of up to 5 characters to be used as the first few characters of the name of the temporary file.

`tempnam` uses `malloc`(3C) to get space for the constructed filename and returns a pointer to this area. Thus, any pointer value returned from `tempnam` may serve as an argument to *free* (see `malloc`(3C)). If `tempnam` cannot return the expected result for any reason (i.e., `malloc` failed or attempts to find an appropriate directory were unsuccessful), a NULL pointer will be returned.

NOTES

These functions generate a different filename each time they are called.

Files created using these functions and either `fopen`(3S) or `creat`(2) are temporary only in the sense that they reside in a directory intended for temporary use and their names are unique. It is the user's responsibility to use `unlink`(2) to remove the file when its use is ended.

SEE ALSO

`creat`(2), `unlink`(2), `fopen`(3S), `malloc`(3C), `mktemp`(3C), `tmpfile`(3S).

BUGS

If called more than 17,576 times in a single process, `tmpnam` and `tempnam` will start recycling previously used names.

Between the time a filename is created and the file is opened, it is possible for some other process to create a file with the same name. This can never happen if that other process is using `tmpnam`, `tempnam`, or `mktemp`(3C) and the filenames are chosen carefully to avoid duplication by other means.

NAME

sin, cos, tan, asin, acos, atan, atan2 — trigonometric functions

SYNOPSIS

```
#include <math.h>

double sin(x)
double x;

double cos(x)
double x;

double tan(x)
double x;

double asin(x)
double x;

double acos(x)
double x;

double atan(x)
double x;

double atan2(y, x)
double x, y;
```

DESCRIPTION

`sin`, `cos`, and `tan` return, respectively, the sine, cosine, and tangent of their argument, which is in radians.

`asin` returns the arcsine of x, in the range $-\pi/2$ to $\pi/2$.

`acos` returns the arccosine of x, in the range 0 to π.

`atan` returns the arctangent of x, in the range $-\pi/2$ to $\pi/2$.

`atan2` returns the arctangent of y/x, in the range $-\pi$ to π, using the signs of both arguments to determine the quadrant of the return value.

RETURN VALUE

`sin`, `cos`, and `tan` lose accuracy when their argument is far from zero. For arguments sufficiently large, these functions return 0 when there would otherwise be a complete loss of significance. In this case a message indicating `TLOSS` error is printed on the standard error output. For less extreme arguments, a `PLOSS` error is generated but no message is printed. In both cases, `errno` is set to `ERANGE`.

If the magnitude of the argument of asin or acos is greater than one, or if both arguments of atan2 are zero, zero is returned and errno is set to EDOM. In addition, a message indicating DOMAIN error is printed on the standard error output.

These error-handling procedures may be changed with the function matherr(3M).

SEE ALSO

matherr(3M).

NAME

tsearch, tfind, tdelete, twalk — manage binary search trees

SYNOPSIS

```
#include <search.h>

char *tsearch(key, rootp, compar)
char *key;
char **rootp;
int(*compar)();

char *tfind(key, rootp, compar);
char *key;
char **rootp;
int(*compar)();

char *tdelete(key, rootp, compar);
char *key;
char **rootp;
int(*compar)();

void twalk(root,action)
char *root;
void(*action)();
```

DESCRIPTION

tsearch, tfind, tdelete, and twalk are routines for manipulating binary search trees. They are generalized from Knuth (6.2.2) Algorithms T and D. All comparisons are done with a user-supplied routine. This routine is called with two arguments, the pointers to the elements being compared. It returns an integer less than, equal to, or greater than 0, according to whether the first argument is to be considered less than, equal to or greater than the second argument. The comparison function need not compare every byte, so arbitrary data may be contained in the elements in addition to the values being compared.

tsearch is used to build and access the tree. *key* is a pointer to a datum to be accessed or stored. If there is a datum in the tree equal to **key* (the value referenced by *key*), a pointer to this found datum is returned. Otherwise, **key* is inserted, and a pointer to it returned. Only pointers are copied, so the calling routine must store the data. *rootp* points to a variable that points to the root of the tree. A NULL value for the variable referenced by *rootp* denotes an empty tree; in this case, the variable will be set to point to the datum which will be at the root of the new tree.

Like `tsearch`, `tfind` will search for a datum in the tree, returning a pointer to it if found. However, if it is not found, `tfind` will return a NULL pointer. The arguments for `tfind` are the same as for `tsearch`.

`tdelete` deletes a node from a binary search tree. The arguments are the same as for `tsearch`. The variable pointed to by *rootp* will be changed if the deleted node was the root of the tree. `tdelete` returns a pointer to the parent of the deleted node, or a NULL pointer if the node is not found.

`twalk` traverses a binary search tree. *root* is the root of the tree to be traversed. (Any node in a tree may be used as the root for a walk below that node.) *action* is the name of a routine to be invoked at each node. This routine is, in turn, called with three arguments. The first argument is the address of the node being visited. The second argument is a value from an enumeration data type

`typedef enum{`*preorder*`,`*postorder*`,`*endorder*`,`*leaf*`} VISIT;`

(defined in the `<search.h>` header file), depending on whether this is the first, second or third time that the node has been visited (during a depth-first, left-to-right traversal of the tree), or whether the node is a leaf. The third argument is the level of the node in the tree, with the root being level zero.

The pointers to the key and the root of the tree should be of type pointer-to-element, and cast to type pointer-to-character. Similarly, although declared as type pointer-to-character, the value returned should be cast into type pointer-to-element.

EXAMPLES

The following code reads in strings and stores structures containing a pointer to each string and a count of its length. It then walks the tree, printing out the stored strings and their lengths in alphabetical order.

```
#include <search.h>
#include <stdio.h>

struct node {                   /*pointers to these are
        char * string;            stored in the tree*/
        int length;
};
```

```
char string_space[10000]; /*space to store
                                       strings*/
struct node nodes[500];    /*nodes to store*/
struct node *root = NULL; /*this points to the
                                       root*/

main( )
{
    char *strptr = string_space;
    struct node *nodeptr = nodes;
    void print_node( ), twalk( );
    int i = 0, node_compare( );

    while(gets(strptr) != NULL && i++ < 500)   {
        /* set node */
        nodeptr->string = strptr;
        nodeptr->length = strlen(strptr);
        /* put node into the tree */
        (void) tsearch((char *)nodeptr, &root,
                node_compare);
        /* adjust pointers, so we
            don't overwrite tree */
        strptr += nodeptr->length + 1;
        nodeptr++;
    }
    twalk(root, print_node);
}
/*
    This routine compares two nodes, based on an
    alphabetical ordering of the string field.
*/
int
node_compare(node1, node2)
struct node *node1, *node2;
{
    return strcmp(node1->string, node2->string);
}
/*
    This routine prints out a node, the
    first time twalk encounters it.
*/
void
```

```
print_node(node, order, level)
struct node **node;
VISIT order;
int level;
{
    if (order == preorder || order == leaf)   {
        (void)printf("string = %20s,  length = %d\n",
            (*node)->string, (*node)->length);
    }
}
```

RETURN VALUE

A NULL pointer is returned by `tsearch` if there is not enough space available to create a new node.

A NULL pointer is returned by `tsearch`, `tfind` and `tdelete` if *rootp* is NULL on entry.

If the datum is found, both `tsearch` and `tfind` return a pointer to it. If not, `tfind` returns NULL, and `tsearch` returns a pointer to the inserted item.

SEE ALSO

`bsearch`(3C), `hsearch`(3C), `lsearch`(3C).

WARNINGS

The *root* argument to `twalk` is one level of indirection less than the *rootp* arguments to `tsearch` and `tdelete`.
There are two nomenclatures used to refer to the order in which tree nodes are visited. `tsearch` uses preorder, postorder and endorder to respectively refer to visting a node before any of its children, after its left child and before its right, and after both its children. The alternate nomenclature uses preorder, inorder and postorder to refer to the same visits, which could result in some confusion over the meaning of postorder.

BUGS

If the calling function alters the pointer to the root, results are unpredictable.

NAME

`ttyname`, `isatty` — find name of a terminal

SYNOPSIS

`char *ttyname` (*fildes*)
`int` *fildes;*

`int isatty` (*fildes*)
`int` *fildes;*

DESCRIPTION

`ttyname` returns a pointer to a string containing the null-terminated pathname of the terminal device associated with file descriptor *fildes.*

RETURN VALUE

`ttyname` returns a NULL pointer if *fildes* does not describe a terminal device in directory `/dev`.

`isatty` returns 1 if *fildes* is associated with a terminal device; otherwise, it returns 0.

FILES

`/dev/*`

BUGS

The return value points to static data whose content is overwritten by each call.

NAME

ttyslot — find the slot in the utmp file of the current user

SYNOPSIS

```
int ttyslot()
```

DESCRIPTION

ttyslot returns the index of the current user's entry in the /etc/utmp file. This is accomplished by scanning the file /etc/inittab for the name of the terminal device associated with the standard input, the standard output, or the error output (0, 1, or 2).

SEE ALSO

getut(3C), ttyname(3C).

FILES

/etc/inittab
/etc/utmp

RETURN VALUE

A value of 0 is returned if an error is encountered while searching for the terminal name or if none of the above file descriptors is associated with a terminal device.

NAME

umount — unmount a file system

SYNOPSIS

```
int umount(spec)
char *spec;
```

DESCRIPTION

umount requests that a previously mounted file system contained on the block special device identified by *spec* be unmounted. *spec* is a pointer to a path name. After unmounting the file system, the directory upon which the file system was mounted reverts to its ordinary interpretation.

umount may be invoked only by the superuser.

ERRORS

umount will fail if one or more of the following are true:

[EPERM]	The process's effective user ID is not superuser.
[ENXIO]	*spec* does not exist.
[ENOTBLK]	*spec* is not a block special device.
[EINVAL]	*spec* is not mounted.
[EBUSY]	A file on *spec* is busy.
[EFAULT]	*spec* points to an illegal address.

RETURN VALUE

Upon successful completion a value of 0 is returned. Otherwise, a value of −1 is returned and errno is set to indicate the error.

SEE ALSO

fsmount(2), unmount(2), mount(3).

NAME

ungetc — push character back into input stream

SYNOPSIS

```
#include <stdio.h>

int ungetc(c, stream)
char c;
FILE *stream;
```

DESCRIPTION

ungetc inserts the character *c* into the buffer associated with an input *stream*. That character, *c*, will be returned by the next getc call on that *stream*. ungetc returns *c* and leaves the file *stream* unchanged.

One character of pushback is guaranteed provided something has been read from the stream and the stream is actually buffered. In the case that *stream* is stdin, one character may be pushed back onto the buffer without a previous read statement.

If *c* equals EOF, ungetc does nothing to the buffer and returns EOF.

fseek(3S) erases all memory of inserted characters.

RETURN VALUE

ungetc returns EOF if it can't insert the character.

SEE ALSO

fseek(3S), getc(3S), setbuf(3S).

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` will return the next argument in the list referenced 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.

EXAMPLES

This example is a possible implementation of `execl`(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).

BUGS

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

`vprintf`, `vfprintf`, `vsprintf` — format and output data from a variable-length argument list

SYNOPSIS

```
#include <stdio.h>
#include <varargs.h>

int vprintf(format, ap)
char *format;
va_list ap;

int vfprintf(stream, format, ap)
FILE *stream;
char *format;
va_list ap;

int vsprintf(s, format, ap)
char *s, *format;
va_list ap;
```

DESCRIPTION

`vprintf`, `vfprintf`, and `vsprintf` are the same as `printf`, `fprintf`, and `sprintf` respectively, except that instead of being called with a variable number of arguments, they are called with an argument list as defined by `varargs`(5).

EXAMPLES

The following demonstrates how `vfprintf` could be used to write an error routine.

```
#include <stdio.h>
#include <varargs.h>
          .
          .
          .
/*
 *   error should be called like
 *      error(function_name, format, arg1, arg2...);
 */
/*VARARGS0*/
void
error(va_alist)
/* Note that the function_name and format arguments
 * cannot be separately declared because of the
 */definition of varargs.
va_dcl
{
   va_list args;
   char *fmt;
```

```
        va_start(args);
        /* print out name of function causing error */
        (void)fprintf(stderr, "ERROR in %s: ",
                           va_arg(args, char *));
        fmt = va_arg(args, char *);
        /* print out remainder of message */
        (void)vfprintf(fmt, args);
        va_end(args);
        (void)abort( );
    }
```

SEE ALSO

varargs(5).

NAME

xdr — library routines for external data representation

DESCRIPTION

These routines allow C programmers to describe arbitrary data structures in a machine-independent fashion. Data for remote procedure calls are transmitted using these routines.

FUNCTIONS

`xdr_array()`	translate arrays to/from external representation
`xdr_bool()`	translate Booleans to/from external representation
`xdr_bytes()`	translate counted byte strings to/from external representation
`xdr_destroy()`	destroy XDR stream and free associated memory
`xdr_double()`	translate double precision to/from external representation
`xdr_enum()`	translate enumerations to/from external representation
`xdr_float()`	translate floating point to/from external representation
`xdr_getpos()`	return current position in XDR stream
`xdr_inline()`	invoke the in-line routines associated with XDR stream
`xdr_int()`	translate integers to/from external representation
`xdr_long()`	translate long integers to/from external representation
`xdr_opaque()`	translate fixed-size opaque data to/from external representation
`xdr_reference()`	chase pointers within structures
`xdr_setpos()`	change current position in XDR stream
`xdr_short()`	translate short integers to/from external representation
`xdr_string()`	translate null-terminated strings to/from external representation
`xdr_u_int()`	translate unsigned integers to/from external representation

`xdr_u_long()`	translate unsigned long integers to/from external representation
`xdr_u_short()`	translate unsigned short integers to/from external representation
`xdr_union()`	translate discriminated unions to/from external representation
`xdr_void()`	always return one (1)
`xdr_wrapstring()`	package RPC routine for XDR routine, or vice-versa
`xdrmem_create()`	initialize an XDR stream
`xdrrec_create()`	initialize an XDR stream with record boundaries
`xdrrec_endofrecord()`	mark XDR record stream with an end-of-record
`xdrrec_eof()`	mark XDR record stream with an end-of-file
`xdrrec_skiprecord()`	skip remaining record in XDR record stream
`xdrstdio_create()`	initialize an XDR stream as standard I/O FILE stream

SEE ALSO

A/UX Network Applications Programming.

NAME

yp_bind, yp_unbind, yp_get_default_domain, yp_match, yp_first, yp_next, yp_all, yp_order, yp_master, yperr_string, ypprot_err — yellow pages client interface

SYNOPSIS

```
#include <rpcsvc/ypclnt.h>

yp_bind(indomain);
char *indomain;

void yp_unbind(indomain)
char *indomain;

yp_get_default_domain(outdomain);
char **outdomain;

yp_match(indomain, inmap, inkey, inkeylen, outval,
outvallen)
char *indomain;
char *inmap;
char *inkey;
int inkeylen;
char **outval;
int *outvallen;

yp_first(indomain, inmap, outkey, outkeylen, outval,
outvallen)
char *indomain;
char *inmap;
char **outkey;
int *outkeylen;
char **outval;
int *outvallen;

yp_next(indomain, inmap, inkey, inkeylen, outkey,
outkeylen, outval, outvallen);
char *indomain;
char *inmap;
char *inkey;
int inkeylen;
char **outkey;
int *outkeylen;
char **outval;
int *outvallen;
```

```
yp_all(indomain, inmap, incallback);
char *indomain;
char *inmap;
struct ypall_callback incallback;

yp_order(indomain, inmap, outorder);
char *indomain;
char *inmap;
int *outorder;

yp_master(indomain, inmap, outname);
char *indomain;
char *inmap;
char **outname;

char *yperr_string(incode)
int incode;

ypprot_err(incode)
unsigned int incode;
```

DESCRIPTION

This package of functions provides an interface to the yellow pages (YP) network lookup service. The package can be loaded from the standard library /lib/libc.a. Refer to ypfiles(4) and ypserv(1M) for an overview of the yellow pages, including the definitions of *map* and *domain*, and a description of the various servers, databases, and commands that comprise the YP.

All input parameters names begin with "in". Output parameters begin with "out". Output parameters of type "char **" should be addresses of uninitialized character pointers. Memory is allocated by the YP client package using malloc(3), and may be freed if the user code has no continuing need for it. For each *outkey* and *outval*, two extra bytes of memory are allocated at the end that contain NEWLINE and NULL, respectively, but these two bytes are not reflected in *outkeylen* or *outvallen*.

indomain and *inmap* strings must be non-null and null-terminated. String parameters which are accompanied by a count parameter may not be null, but may point to null strings, with the count parameter indicating this. Counted strings need not be null-terminated.

All functions in this package of type "int" return 0 if they succeed, and a failure code (YPERR_*xxxx*) otherwise. Failure codes are described under ERRORS below.

The YP lookup calls require a map name and a domain name, at minimum. It is assumed that the client process knows the name of the map of interest. Client processes should fetch the node's default domain by calling `yp_get_default_domain()`, and use the returned *outdomain* as the *indomain* parameter to successive YP calls.

To use the YP services, the client process must be "bound" to a YP server that serves the appropriate domain using `yp_bind`. Binding need not be done explicitly by user code; this is done automatically whenever a YP lookup function is called. `yp_bind` can be called directly for processes that make use of a backup strategy (e.g., a local file) in cases when YP services are not available.

Each binding allocates (uses up) one client process socket descriptor; each bound domain costs one socket descriptor. However, multiple requests to the same domain use that same descriptor. `yp_unbind()` is available at the client interface for processes that explicitly manage their socket descriptors while accessing multiple domains. The call to `yp_unbind()` make the domain "unbound," and free all per-process and per-node resources used to bind it.

If an RPC failure results upon use of a binding, that domain will be unbound automatically. At that point, the `ypclnt` layer will retry forever or until the operation succeeds, provided that `ypbind` is running, and either

> the client process can't bind a server for the proper domain, or
>
> RPC requests to the server fail.

If an error is not RPC-related, or if `ypbind` is not running, or if a bound `ypserv` process returns any answer (success or failure), the `ypclnt` layer will return control to the user code, either with an error code, or a success code and any results.

`yp_match` returns the value associated with a passed key. This key must be exact; no pattern matching is available.

`yp_first` returns the first key-value pair from the named map in the named domain.

yp_next() returns the next key-value pair in a named map. The *inkey* parameter should be the *outkey* returned from an initial call to yp_first() (to get the second key-value pair) or the one returned from the *n*th call to yp_next() (to get the *n*th + second key-value pair).

The concept of first (and, for that matter, of next) is particular to the structure of the YP map being processing; there is no relation in retrieval order to either the lexical order within any original (non-YP) data base, or to any obvious numerical sorting order on the keys, values, or key-value pairs. The only ordering guarantee made is that if the yp_first() function is called on a particular map, and then the yp_next() function is repeatedly called on the same map at the same server until the call fails with a reason of YPERR_NOMORE, every entry in the data base will be seen exactly once. Further, if the same sequence of operations is performed on the same map at the same server, the entries will be seen in the same order.

Under conditions of heavy server load or server failure, it is possible for the domain to become unbound, then bound once again (perhaps to a different server) while a client is running. This can cause a break in one of the enumeration rules; specific entries may be seen twice by the client, or not at all. This approach protects the client from error messages that would otherwise be returned in the midst of the enumeration. The next paragraph describes a better solution to enumerating all entries in a map.

yp_all provides a way to transfer an entire map from server to client in a single request using TCP (rather than UDP as with other functions in this package). The entire transaction take place as a single RPC request and response. You can use yp_all just like any other YP procedure, identify the map in the normal manner, and supply the name of a function which will be called to process each key-value pair within the map. You return from the call to yp_all only when the transaction is completed (successfully or unsuccessfully), or your "foreach" function decides that it doesn't want to see any more key-value pairs.

The third parameter to yp_all is

```
struct ypall_callback *incallback {
int (*foreach)();
char *data;
};
```

The function foreach is called

```
foreach(instatus, inkey, inkeylen, inval, invallen, indata);
int instatus;
char *inkey;
int inkeylen;
char *inval;
int invallen;
char *indata;
```

The *instatus* parameter will hold one of the return status values defined in <rpcsvc/yp_prot.h>; either YP_TRUE or an error code. (See ypprot_err, below, for a function which converts a YP protocol error code to a ypclnt layer error code.)

The key and value parameters are somewhat different than defined in the synopsis section above. First, the memory pointed to by the *inkey* and *inval* parameters is private to the yp_all function, and is overwritten with the arrival of each new key-value pair. It is the responsibility of the foreach function to do something useful with the contents of that memory, but it does not own the memory itself. Key and value objects presented to the foreach function look exactly as they do in the server's map; if they were not newline-terminated or null-terminated in the map, they won't be here either.

The *indata* parameter is the contents of the *incallback->data* element passed to yp_all. The *data* element of the callback structure may be used to share state information between the foreach function and the mainline code. Its use is optional, and no part of the YP client package inspects its contents; cast it to something useful, or ignore it as you see fit.

The foreach function is a Boolean. It should return zero to indicate that it wants to be called again for further received key-value pairs, or non-zero to stop the flow of key-value pairs. If foreach returns a non-zero value, it is not called again; the functional value of yp_all is then 0.

yp_order returns the order number for a map.

yp_master returns the machine name of the master YP server for a map.

yperr_string returns a pointer to an error message string that is null-terminated but contains no period or newline.

ypprot_err takes a YP protocol error code as input, and returns a ypclnt layer error code, which may be used in turn as an input to yperr_string.

ERRORS

All integer functions return 0 if the requested operation is successful, or one of the following errors if the operation fails.

```
#define YPERR_BADARGS 1  /* args to function are bad */
#define YPERR_RPC     2  /* RPC failure - domain has
                            been unbound */
#define YPERR_DOMAIN  3  /* can't bind to server on this
                            domain */
#define YPERR_MAP     4  /* no such map in server's
                            domain */
#define YPERR_KEY     5  /* no such key in map */
#define YPERR_YPERR   6  /* internal yp server or
                            client error */
#define YPERR_RESRC   7  /* resource allocation
                            failure */
#define YPERR_NOMORE  8  /* no more records in map
                            database */
#define YPERR_PMAP    9  /* can't communicate with
                            portmapper */
#define YPERR_YPBIND  10 /* can't communicate with
                            ypbind */
#define YPERR_YPSERV  11 /* can't communicate with
                            ypserv */
#define YPERR_NODOM   12 /* local domain name not set */
```

FILES

/usr/include/rpcsvc/ypclnt.h
/usr/include/rpcsvc/yp_prot.h

SEE ALSO

ypserv(1M), ypfiles(4).

NAME

yppasswd — update user password in yellow pages

SYNOPSIS

```
#include <rpcsvc/yppasswd.h>

yppasswd(oldpass, newpw)
char *oldpass;
struct passwd *newpw;
```

DESCRIPTION

If *oldpass* is indeed the old user password, this routine replaces the password entry with *newpw*. It returns 0 if successful.

RPC INFO

Program number: `YPPASSWDPROG`

`xdr` routines:

```
xdr_ppasswd(xdrs, yp)
    XDR *xdrs;
    struct yppasswd *yp;
xdr_yppasswd(xdrs, pw)
    XDR *xdrs;
    struct passwd *pw;
```

Procs:

`YPPASSWDPROC_UPDATE`

Takes the structure `yppasswd` as an argument; returns integer. Same behavior as the `yppasswd()` wrapper. Uses UNIX authentication.

Versions:

`YPPASSWDVERS_ORIG`

Structures:

```
struct yppasswd {
  char *oldpass; /* old (unencrypted) pw */
  struct passwd newpw; /* new pw structure */
};
```

SEE ALSO

yppasswd(1), yppasswdd(1M).

NAME

zip_getmyzone, zip_getzonelist, zip_getlocalzones — AppleTalk Zone Information Protocol (ZIP) interface

SYNOPSIS

```
#include <at/appletalk.h>
#include <at/zip.h>
```

cc [*flags*] *files* -lat [*libraries*]

int zip_getmyzone(*zone*) at_nvestr_t **zone;*

int zip_getzonelist(*start*, *zones*) int *start;* at_nvestr_t **zones*[];

int zip_getlocalzones(*start*, *zones*) int *start;* at_nvestr_t **zones*[];

DESCRIPTION

The ZIP interface provides applications with access to the AppleTalk Zone Information Protocol operations.

The zip_getmyzone routine obtains the zone name for the local network. In the case of LocalTalk, this involves sending a ZIP request to a local bridge to get the zone name of the default network. In the case of EtherTalk, the request is completed on the node itself. The parameters are

zone A pointer to the zone name. The zone string is defined by the following structure (see <at/nbp.h>):

```
typedef struct  at_nvestr {
        u_char  len;
        u_char  str[NBP_NVE_STR_SIZE];
} at_nvestr_t;
```

len The size of the string in bytes.

str The zone name.

This routine returns 0 upon success.

The zip_getzonelist routine obtains a complete list of all the zone names defined in the internet. This routine sends a ZIP request to a bridge for the list of zone names in the internet. The list is placed in the supplied buffer as concatenated at_nvestr_t structures. The parameters are *start* The starting index for the get zone list request. The start index is the value of the index at which to start including zone names in the response. It is used to obtain a zone list that may not fit into one ATP response packet. The start index should initially be 1. While `zip_getzonelist` returns a value greater than 0, the caller must reissue `zip_getzonelist` calls to get more zone names from the bridge, specifying a start index of the previous start index plus the previous return value of `zip_getzonelist`.

buf A buffer to hold this list of zone names. Each zone name is an `at_nvestr_t` structure. The size of this buffer (in bytes) must be at least `ATP_DATA_SIZE`.

Upon successful completion, this routine returns the number of zone names in the list.

When all zones in the bridge's Zone Information Table have been returned, this routine returns 0.

The use and behavior of the `zip_getlocalzones` routine are the same as for `zip_getzonelist`, except that the former returns the list of zones on the local EtherTalk cable rather than all the zones on the internet. On LocalTalk, `zip_getlocalzones` returns only the current zone name.

DIAGNOSTICS

All routines return −1 on error, with a detailed error code stored in `errno`:

`[EINVAL]`	A parameter is invalid.
`[ENETUNREACH]`	A bridge node could not be found to process the request.

Routines also return any error codes returned by the underlying ATP or DDP layers.

SEE ALSO

`ddp`(3N), `atp`(3N), *Inside AppleTalk;* "AppleTalk Programming Guide," in *A/UX Network Applications Programming*.

WARNINGS

The returned zone strings are not NULL-terminated.

Table of Contents

Section 4: File Formats

intro(4) .... introduction to file formats
a.out(4) .... common assembler and link editor output
acct(4) .... per-process accounting file format
afm(4) .... Adobe POSTSCRIPT font metrics file format
aliases(4) .... aliases file for sendmail
altblk(4) .... alternate block information for bad block handling
aouthdr(4) .... a.out header for common object files
appletalkrc(4) .... AppleTalk® network configuration file
ar(4) .... common archive file format
bzb(4) .... format of Block Zero Blocks
cml(4) .... configuration master list format
core(4) .... format of core image file
cpio(4) .... format of cpio archive
dialup(4) .... modem escape sequence file
dir(4) .... format of System V directories
disktab(4) .... disk description file
dpme(4) .... format of disk partition map entries
dump.bsd(4) .... format of a file system dump
errfile(4) .... error-log file format
ethers(4) .... Ethernet address to hostname database or YP domain
exports(4) .... NFS file systems being exported
filehdr(4) .... file header for common object files
finstallrc(4) .... finstall default configuration file
fs(4) .... file systems
fspec(4) .... syntax for format lines for newform
fstab(4) .... static information about file systems
fstypes(4) .... name-mapping information for file systems
gettydefs(4) .... speed and terminal settings used by getty
group(4) .... group file
HOSTNAME(4) .... hostname and domainname database
hosts(4) .... host name database
hosts.equiv(4) .... list of trusted hosts
inittab(4) .... script for the init process
inode(4) .... format of a System V inode
ioctl.syscon(4) .... console terminal settings file
issue(4) .... issue identification file
iwmap(4) .... format of iwprep(1) character map description files

linenum(4) .... line number entries in a common object file
magic(4) .... magic number file for file command
master(4) .... master kernel configuration files
mtab(4) .... mounted file system table
NETADDRS(4) .... network address database
netgroup(4) .... list of network groups
networks(4N) .... network name database
passwd(4) .... password file
phones(4) .... remote host telephone number database
plot(4) .... graphics interface
postscript(4) .... POSTSCRIPT print file format
printcap(4) .... printer-capability database
profile(4) .... setting up an environment at login time
protocols(4N) .... protocol name database
ptab(4) .... partition table file
rcsfile(4) .... format of an RCS file
reloc(4) .... relocation information for a common object file
remote(4) .... remote host description file
resolver(4) .... resolver configuration file
rhosts(4N) .... trusted hosts file format
rmtab(4) .... remotely mounted file system table
rpc(4) .... RPC program number database
sccsfile(4) .... format of an SCCS file
scnhdr(4) .... section header for a common object file
servers(4) .... Internet server database
services(4N) .... service name database
slip.config(4) .... list of slip interfaces supported by a slip server
slip.hosts(4) .... map user names to host addresses of slip client
slip.user(4) .... user file created by mkslipuser
svfs(4) .... format of a System V system volume
syms(4) .... common object file symbol table format
tar(4) .... format of tar header
term(4) .... format of compiled term file
termcap(4) .... terminal capability database
terminfo(4) .... terminal capability database
ttytype(4) .... database of terminal types by port
tzfile(4) .... time-zone information
ufs(4) .... format of a UFS file-system volume
utmp(4) .... utmp and wtmp entry formats
wtmp(4) .... see utmp(4)
ypfiles(4) .... the Yellow Pages database and directory structure

NAME

`intro` — introduction to file formats

DESCRIPTION

This section outlines the formats of various files. The C `struct` declarations for the file formats are given where applicable. Usually, these structures can be found in the directories `/usr/include` or `/usr/include/sys`.

NAME

`a.out` — common assembler and link editor output

DESCRIPTION

`a.out` is the output file from the assembler `as`(1) and the link editor `ld`(1). `a.out` can be executed on the target machine if there were no errors in assembling or linking and no unresolved external references.

The object file format supports user-defined sections and contains extensive information for symbolic software testing. A common object file consists of a file header, an optional `aout` header, a table of section headers, relocation information, (optional) line numbers, and a symbol table. The order is given below.

File header.
Optional `aout` 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 four sections (relocation, line numbers, symbol table, and string table) may be missing if the program was linked with the `-s` option of `ld`(1) or if the symbol table and relocation bits were removed by `strip`(1). Also note that if the program was linked without the `-r` option, the relocation information will be absent. The string table exists only if necessary.

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 data, the latter actually being initialized to all 0's), and a stack. The text segment begins at location 0 in the core image; the header is not loaded. If the magic number (the first field in the optional aout header) is 407 (octal), it indicates that the text segment is not to be write-protected or shared, so the data segment will be contiguous with the text segment. If the magic number is 410 (octal), the data segment begins at the next segment boundary following the text segment, and the text segment is not writable by the program. If other processes are executing the same a.out file, they will share a single text segment.

On the Macintosh II with A/UX the stack begins at the end of memory and grows toward lower addresses. The stack is automatically extended as required. The data segment is extended only as requested by the brk(2) and sbrk(2) system calls.

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, the storage class of the symbol-table entry for that word will be marked as an "external symbol", and the section number will be set to 0. 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.

See aouthdr(4), filehdr(4), linenum(4), scnhdr(4), reloc(4), and syms(4) for descriptions of the individuals parts. Every section created by as(1) contains a multiple-of-four number of bytes; directives to ld(1) can create a section with an odd number of bytes.

SEE ALSO

as(1), cc(1), ld(1), ldfcn(3X), aouthdr(4), filehdr(4), linenum(4), reloc(4), scnhdr(4), syms(4).

NAME

acct — per-process accounting file format

SYNOPSIS

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

DESCRIPTION

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

```
typedef ushort comp_t;  /* floating point */
                        /* 13-bit fraction, 3-bit
                           exponent */

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

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

#define  AFORK   01     /* has executed fork-no exec */
#define  ASU     02     /* used superuser privileges */
#define  ACCTF   0300   /* record type: 00 = acct */
```

In `ac_flag`, the `AFORK` flag is turned on by each `fork(2)` and turned off by an `exec(2)`. The `ac_comm` field is inherited from the parent process and is reset by any `exec`. Each time the system charges the process with a clock tick, it also adds the current process size to `ac_mem`, computed as follows:

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

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

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

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

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

SEE ALSO

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

BUGS

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

NAME

afm — Adobe POSTSCRIPT font metrics file format

DESCRIPTION

afm files are a standard interchange format for communicating POSTSCRIPT font metric information to people and programs. The format is ASCII encoded (for both human and machine readability), machine independent, extensible, simple to parse, and simple to generate. afm files are available for all of Adobe Systems' POSTSCRIPT fonts.

While somewhat verbose, the format is intended to be easily parsed, with the ability for applications to quickly skip over items that are not of interest. It should be possible to create simple line-oriented parsing programs, or tools based on awk(1) or sed(1).

Each afm file contains the information for only one font face. The file begins with global information about the font, followed by sections with character metrics. The file format is line-oriented, each line beginning with a property (key) name, followed by the values for that property. The general idea is to give key-value tuples (much like in a POSTSCRIPT font dictionary).

The format is:

key [*value value* ...]

Key names are case-sensitive. All keys beginning with a capital letter are reserved by Adobe Systems. The standard keys are detailed below, but other keys should be allowed and safely ignored by programs not recognizing them. All standard keys begin with a capital letter. User-defined nonstandard entries should begin with a lowercase letter.

The file begins with the line:

StartFontMetrics *version*

The version described here is 1.0. The last line of the file is:

EndFontMetrics

The following global font keys are defined. Many of them are defined as in the top level or FontInfo subdictionary of a POSTSCRIPT font dictionary; their meanings are described in Appendix A of the *POSTSCRIPT Language Manual*. All numeric values are in the (1000 unit per em) character coordinate system.

`FontName` *string*
: The name of the font as presented to the POSTSCRIPT `findfont` operator.

`FullName` *string*
: The "print name" of the font.

`FamilyName` *string*
: The font family name.

`Weight` *string*
: The weight of the font.

`ItalicAngle` *real*
: The angle (in degrees counter clockwise from the vertical) of the dominant staffs of the font.

`IsFixedPitch` *boolean*
: Indicates monospaced (typewriter) fonts.

`FontBBox` *llx lly urx ury*
: Four integers giving the lower left corner and the upper right corner of the font bounding box.

> *Note:* The bounding box given here is that of the flattened paths, not of the Bezier curve descriptions.

`UnderlinePosition` *number*
: The position (from the baseline) to place an underline.

`UnderlineThickness` *number*
: Thickness of an underline stroke.

`Version` *string*
: Font version identifier.

`Notice` *string*
: Font name trademark or copyright notice.

`Comment` *string*
: Comment strings may be ignored.

`EncodingScheme` *string*
: a string indicating the default encoding vector for this font. The most common one is `AdobeStandardEncoding`. Special fonts may simply state `FontSpecific`. In the future, other schemes may be employed.

`CapHeight` *number*
: Top of capital H.

`XHeight` *number*	Top of lowercase x.
`Ascender` *number*	Top of lowercase d.
`Descender` *number*	Bottom of lowercase p.

The individual character metrics are surrounded with the lines `StartCharMetrics` and `EndCharMetrics` and consist of a list of keys and values separated by semicolons. The characters are sorted (numeric ascending) by character code. Unencoded characters follow all of the encoded ones and are distinguished by having character code −1. Each character gets one line of description. Standard keys are:

`C` *number*	decimal value of default POSTSCRIPT character code (−1 if unencoded).
`WX` *width-x*	Character width in *x* (*y* is 0).
`W` *width-x width-y*	Character width vector.
`N` *name*	POSTSCRIPT character name.
`B` *llx lly urx ury*	The character bounding box.
`L` *successor ligature*	A ligature sequence. The current character may join with the character named *successor* to form the character named *ligature*. Note that characters may have more than one such entry.

Most western language fonts have `WX` entries rather than `W` ones. Note that keys are one letter for brevity. Here too, the set is extensible, with unknown entries ignored. (This leaves room for addition of new information, for example.) A future revision of this format will have a specification for kerning information.

FILES

`/usr/lib/ps/*.afm` AFM files in the TRANSCRIPT distribution.

SEE ALSO

`awk`(1), `sed`(1).

NAME
aliases — aliases file for sendmail

SYNOPSIS
/usr/lib/aliases

DESCRIPTION
This file describes user ID aliases that /etc/sendmail uses. It is a series of lines of the form

name : *addr1* , *addr2* , . . . *addrn*

The *name* is the name to alias, and the *addr*'s are the addresses to send the message to. Lines beginning with white space are continuation lines. Lines beginning with # are comments.

Aliasing occurs only on local names. Loops cannot occur, since no message is sent to any person more than once.

FILES
/usr/lib/aliases

SEE ALSO
sendmail(1M).

NAME

altblk — alternate block information for bad block handling

SYNOPSIS

```
#include <sys/types.h>
#include <apple/abm.h>
```

DESCRIPTION

abm is the data structure used by A/UX disk device drivers to handle bad blocks for disk partitions that support alternate block bad block handling. The abm structure can be retrieved through an ioctl(2) with a request of GD_GETABM. The actual contents of the alternate block map can be retrieved via the abmi structure through an ioctl(2) with a request of GD_GETMAP. The abmi structure is described in gd(7). The format of the abm structure is:

```
struct abm            /* altblk map info stored in bzb */
{
    int     abm_size;      /* size of map (bytes) */
    int     abm_ents;      /* number of used entries
                              (bytes) */
    daddr_t abm_start;     /* start of altblk map
                              (phys blk num) */
};
typedef  struct abm ABM;

#define ABM_ENTSIZ (sizeof(long)) /* size of each
                                     map entry */
#define NO_ALTMAP  ((daddr_t) 0)  /* value of abm_off
                                     field if no map */

#define   ABM_FREE    -1   /* block not used */
#define   ABM_BADBLK  -2   /* block is bad */
#define   ABM_ABM     -3   /* part of AltBlkMap */
#define   ABM_MAXVAL  -16  /* last reserved map value */
```

Normally the alternate block area, that area between the end of the logical partition and the end of the physical partition, will (optionally) contain an alternate block map and alternate block data blocks for alternate block handling. The alternate block map resides anywhere in the alternate block area, in a contiguous set of blocks. The format of the alternate block map is an array of long integers. Each indexed location in the array corresponds to a potential alternate block in the alternate block area. A location in the alternate block array (map) may either contain the number of a block in the logical partition of the disk partition which will be remapped, or it may contain a flag.

The currently recognized flag values are ABM_FREE for available blocks, ABM_BADBLK if the free block is bad, and ABM_ABM to indicate that the block is allocated to the alternate block map. Flag values with in the range of ABM_ABM and ABM_MAXVAL are reserved for future use.

Alternate block mapping may be disabled through an ioctl(2) with the request GD_ALTBLK. A bad block may be alternate blocked through an ioctl(2) with the request GD_MKBAD.

FIELD DESCRIPTIONS

abm_size
: This field contains the size of the alternate block map as measured in bytes. This value should always be evenly divisible by ABM_ENTSIZ. The value of this field should be consulted when requesting the contents of the alternate block map through an ioctl(2).

abm_ents
: The value of this field represents the byte offset of the next available entry in the alternate block map as measured from the beginning of the map. This field is maintained by the device driver.

abm_start
: The value of this field is set to NO_ALTMAP to indicate that there is no alternate block map for the corresponding partition. If the value of this field is not set to NO_ALTMAP, then the value is the physical block number (relative to the start of the physical partition) of the first block of the alternate block map.

SEE ALSO

badblk(1M), dp(1M), bzb(4), gd(7).

NAME

aouthdr — a.out header for common object files

SYNOPSIS

`#include <aouthdr.h>`

DESCRIPTION

aouthdr is an optional a.out header for common object files. The C structure follows:

```
/*
 * static char ID_aouth[] = "@(#)aouthdr.h  2.1    ";
 */

typedef struct  aouthdr {
        short   magic;    /* see magic.h  */
        short   vstamp;   /* version stamp */
        long    tsize;    /* text size in bytes,
                             padded to FW bdry */
        long    dsize;    /* initialized data " "  */
        long    bsize;    /* uninitialized data " " */
#ifdef u3b
        long   dum1;
        long   dum2;      /*Pad to entry point  */
#endif
        long   entry;     /* entry pt.   */
        long   text_start;/* base of text used
                             for this file */
        long   data_start;/* base of data used
                             for this file */
} AOUTHDR;
```

SEE ALSO

a.out(4).

NAME

`appletalkrc` — AppleTalk® network configuration file

DESCRIPTION

`appletalkrc` contains information for configuring an AppleTalk network. The file is created at boot time by the AppleTalk startup routine and is configured for EtherTalk™ by default. The format of the file consists of a list of parameters and values, one per line:

parameter=value

Comments are indicated by a # character and continue until the newline. The following parameters are defined:

`interface`
: The name of the default AppleTalk interface. The *value* for this parameter can be either `ethertalk0` or `localtalk0`.

`ethernet`
: The name of the hardware interface to be associated with the EtherTalk interface. The *value* for this parameter is a string such as `ae0`.

EXAMPLES

This is the default `appletalkrc` file created by the AppleTalk startup routine for a system with one EtherTalk card:

```
# AppleTalk configuration file

interface=  ethertalk0

ethernet= ae0
```

FILES

`/etc/appletalkrc`
`/etc/startup.d`
`/etc/newunix`

SEE ALSO

`appletalk`(1M), `newunix`(1M).

"Installing and Administering AppleTalk," in *A/UX Network System Administration*; *Inside AppleTalk*; "AppleTalk Programming Guide," in *A/UX Network Applications Programming*.

NAME

ar — common archive file format

DESCRIPTION

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

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 times the number of symbols.
- The name string table. Length: `ar_size` - (4 bytes times (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 corresponds to 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 s option of the ar(1) command before the archive can be used with the link editor ld(1).

NAME

bzb — format of Block Zero Blocks

SYNOPSIS

```
#include <sys/types.h>
#include <apple/types.h>
#include <apple/bzb.h>
```

DESCRIPTION

The Block Zero Block structure occupies the first `sizeof(struct bzb)` bytes of the `dpme_boot_args` field of each A/UX disk partition map entry. This structure contains extra partition identification information that is of interest only to A/UX. The types of data stored in a Block Zero Block include file system identification and status information. The format of a Block Zero Block is

```
struct bzb                          /* block zero block format */
{
        u32     bzb_magic;          /* magic number */
        u8      bzb_cluster;        /* autorecovery cluster
                                       grouping */
        u8      bzb_type;           /* FS type */
        u16     bzb_inode;          /* bad block inode number */
        u16     bzb_root:1,         /* FS is a root FS */
                bzb_usr:1,          /* FS is a usr FS */
                bzb_crit:1,         /* FS is a critical FS */
                bzb_rsrvd:13;       /* reserved for later use */
        u16     bzb_filler;         /* pad bitfield to 32 bits */
        time_t  bzb_tmade;          /* time of FS creation */
        time_t  bzb_tmount;         /* time of last mount */
        time_t  bzb_tumount;        /* time of last umount */
        ABM     bzb_abm;            /* altblk map info */
};
typedef struct bzb BZB;

#define BZBMAGIC ((u32) 0xABADBABE  /* BZB magic number */
#define dpme_bzb dpme_boot_args

/*
** File system types
*/
#define FST     ((u8) 0x1)      /* standard A/UX FS */
#define FSTEFS  ((u8) 0x2)      /* autorecovery FS */
#define FSTSFS  ((u8) 0x3)      /* swap FS */
```

FIELD DESCRIPTIONS

`bzb_magic`
: This field should always contain the magic number BZBMAG-IC. If this field is not set to BZBMAGIC, the information in the Block Zero Block should be treated as invalid.

`bzb_cluster`
: The value of this field determines the autorecovery(8) cluster to which the associated disk partition belongs.

`bzb_type`
: This field identifies the type of A/UX file system corresponding to this Block Zero Block. Examples of A/UX file systems are regular file systems, autorecovery file systems, and swap file systems (for these types, this field's values would be FST, FSTEFS, and FSTSFS, respectively).

`bzb_inode`
: If nonzero, this field contains the number of the inode corresponding to the bad block file in the corresponding partition that will be used for bad blocking. If this field's value is zero, there is no bad block inode/file allocated. This file is made up of blocks that are bad (that is, blocks containing the *contents* of this file are all bad). This keeps the bad blocks out of the free list across fscks. This field is generally used only for file systems that reside on physical disks that lack hardware bad blocking or that support hardware bad blocking but have run out of spare bad blocks. This field is not supported for swap file systems.

  The only supported values for this field are zero and one.

`bzb_root`
: When on, this bit indicates that the file system located on the corresponding partition is a root file system.

`bzb_usr`
: When on, this bit indicates that the file system located on the corresponding partition is a usr file system. If both this field and the bzb_root are on, the file system is a root/usr file system.

`bzb_crit`
: When on, this bit indicates that the file system located on the corresponding partition is a critical file system. A critical file system receives special treatment during the Bad Block por-

tion of `autorecovery`. The swap file system is an example of a critical file system, and therefore, all swap file system Block Zero Blocks should have this field set.

If this bit is on, no attempt will be made to create or use a bad block file for bad block handling.

`bzb_rsrvd`
This field contains bits reserved for later use.

`bzb_filler`
This field is reserved for later use.

`bzb_tmade`
This field contains a time-stamp which indicates when the file system located on the corresponding partition was created. This field's value is the standard A/UX time-stamp value (as returned from `time`(2)). The value of this field can be set and retrieved through calls to `ioctl`(2). See `gd`(7) for more details.

`bzb_tmount`
This field indicates the date the last `mount`(3) (or equivalent routine) call was made on the file system located on the corresponding partition. In some cases, such as on a root file system during startup, this field should be set to the `mount`(3) equivalent date. This field is not updated if a file system is mounted read-only. The value of this field can be set and retrieved through calls to `ioctl`(2). See `gd`(7) for more details.

`bzb_tumount`
This field indicates the date the last `umount`(3) (or equivalent routine) call was made on the file system located on the corresponding partition. In some cases, such as root file system during shutdown, this field should be set to the `umount`(3) equivalent date. This field is not updated if an file system was mounted read-only. The value of this field can be set and retrieved through calls to `ioctl`(2). See `gd`(7) for more details.

`bzb_abm`
This field is the alternate block map structure for the associated partition. See `altblk`(4) for more details about this structure. The value of this field can be retrieved through calls to `ioctl`(2). See `gd`(7) for more details.

SEE ALSO

dp(1M), pname(1M), altblk(4), dpme(4), gd(7), autorecovery(8).

NAME

`cml` — configuration master list format

DESCRIPTION

The Configuration Master List (CML) defines each and every file in the standard A/UX product, and is used to produce and control the A/UX distributions. The CML is also used by `autorecovery`(8), to bring the system up in (minimum) multiuser mode.

The CML files are ASCII text files, containing one record (line) per filename entry, sorted in order by filename. Each record contains multiple tab-separated fields, describing a single file. Each field contains one or more subfields; if more than one, the subfields are separated by colons. The first subfield contains either a filename, a rule for determining the validity of the file, or textual information relating to the file. Additional subfields (if present) contain recognized values associated with the given rule.

No field may be empty; that is, the first subfield must always contain at least one (nonblank) character. To indicate "no rule," the character – is used. Value subfields (that is, subfields past number 1) may be null or missing if they do not apply in the given case. The subfields *must* occur in the specified order, however. Possible additional subfields are given in parentheses after each field name. For example, a partial record might contain

```
- r:m /unix f - <>:100 =>:529799000 u:root...
```

Currently there are 18 recognized fields in a record.

1. *master_rule*
   A string field indicating the master rule for interpreting the validity of this file. The legal rules and their "valid if" conditions are

   `$` If the first character of the *master_rule* field is `$`, the field and subfield delimiters (normally tab and :) are substituted, respectively, by the characters represented by the four hex digits following the `$`; that is, if the first line contains `$407c`, the field and subfield separators will be `@` (hex 40) and `|` (hex 7c). Any changes to these delimiters take effect immediately and remain in effect until the next `$` change. The `$` rule may be used at any point in the CML file where the field and subfield delimiters must be changed.

- `#` signifies a "no-op" condition; the remainder of this record is to be taken as a comment and no calculations are to be performed for any field. The # rule is a way to ignore information for a given file.
- `–` Evaluate, in order, the remaining rules in this record to determine the validity of this file.

2. *autorecovery_rule*[:*autorecovery_value*]
   A string field indicating the autorecovery rule required for interpreting the validity of this file. The legal rules and their "valid if" conditions are

   - `r` This file is required for autorecovery.
   - `–` This file is not required for autorecovery.

   The following value is recognized.

   autorecovery_value
   A string indicating the type of use for which the autorecovery rule applies. The legal value type is

   - `m` Files which are necessary to bring the system up in multiuser mode.

3. *filename*
   A string field containing the fully qualified filename of the file being described. A fully qualified filename starts with a slash and gives the unambiguous placement of the file in the directory hierarchy. (In some cases a filename can not be fully qualified. Any filename not beginning with a slash is assumed to describe a file that may occur in multiple directories (such as `.login`). Such files are not used in autorecovery.)

4. *file_type*
   A string field containing the file type. The legal file types are

   - `d` The file is a directory.
   - `f` The file is a normal file.
   - `b` The file is a block special file.
   - `c` The file is a character special file.
   - `p` The file is a named pipe.
   - `l` The file is a symbolic link.

s The file is a socket.

5. *linked_file_name*
If the file named by *filename* is a symbolic link (*file_type*== l), this field contains the fully qualified pathname of the file to which *filename* is linked. If *filename* does not exist on startup, it is created by linking to *linked_file_name*.

If the file named by *filename* is one of a set of multiple hard links (*file_type*≠ d && line count > 1), this field contains the fully qualified pathname of the aphabetically first file (ASCII sort order) of the set. If a file does not exist on startup and *linked_file_name* == *filename*, it is created by retrieving a copy from the eschatology file system. If a file does not exist on startup and *linked_file_name* ≠ *filename*, it is created by linking to *linked_file_name*.

If there is no *linked_file_name*, this field contains –.

6. *size_rule*[: *size_value_1* : *size_value_2*)
A string field indicating the size rule for interpreting the validity of this file by examining its file length. The legal rules and their ''valid if'' conditions are

<> *size_minimum* ≤ *actual_file_length* ≤ *size_maximum*

== *actual_file_length* = *size_exact*

0= *actual_file_length* = *size_exact* || *actual_file_length* = 0

% *size_exact* – *size_pct*% ≤ *actual_file_length* ≤ *size_exact* + *size_pct*%

– no *size_rule*, hence always true

The following values are recognized.

size_value_1
This is a decimal number which contains the *size_minimum* or *size_exact* depending on the *size_rule* specified. For files which do not have lengths (such as the special files) this value is always 0.

size_value_2
This is a decimal number which contains the *size_maximum* or *size_pct* when required by the *size_rule*. *size_pct* is a decimal percentage (0 < *size_pct* < 100). If the rule is <>, an empty field indicates that there is no set maximum limit.

7. *time_rule*[: *mtime_value*]
   A string field indicating the time rule for interpreting the validity of this file. The legal rules and their "valid if" conditions are

   == *actual_mtime* = *mtime_value*

   => *actual_mtime* ≥ *mtime_value*

   – No time rule, hence always true.

   The following value is recognized.

   mtime_value
      A decimal number containing the appropriate modification time of the file, as required by the *time_rule*.

8. *ownership_rule*[: *file_user* : *file_group*]
   A string field containing the ownership rule for determining the validity of this file. The legal rules and their "valid if" conditions are

   u *actual_user* = *file_user*

   g *actual_group* = *file_group*

   b *actual_user* = *file_user* && *actual_group* = *file_group*

   – No ownership rule, hence always true.

   The following values are recognized.

   file_user
      A string containing the user name.

   file_group
      A string field containing the group name.

9. *permissions_rule*[: *file_mode*]
   A string field containing the permission rule for determining the validity of this file. The legal rules and their "valid if" conditions are

   == *actual_file_mode* = *file_mode*

   – No permissions rule, hence always true.

   The following value is recognized.

   file_mode
      An octal numeric field containing the read/write and other file modes. Note that 0 ≤ *file_mode* ≤ 07777.

10. *major_minor_rule*[:*major_number*:*minor_number*]
    A string field indicating the major/minor rule for interpreting the validity of this file. The *major_minor_rule*, if specified, is ignored *unless* the file is a device (*file_type* = `character` or *file_type* = `block`). The legal rules and their "valid if" conditions are

    ==

    actual_major_number = *major_number* &&
    actual_minor_number = *minor_number*

    – Not a device, hence, always true.

    The following values are recognized.

    major_number
    A decimal number containing the appropriate major device number required by the *major_minor_rule*.

    minor_number
    A decimal number containing the appropriate minor device number required by the *major_minor_rule*.

11. *version_rule*[:*version_value*:*version_minimum*:*version_maximum*]
    A string field indicating the version rule for interpreting the validity of this file by looking for a version number. If a file is made up of several modules, the version number used will be found in the "main" module, as part of a specifically formatted "key version string." The legal rules and their "valid if" conditions are

    <> If version number is present,

    version_minimum ≤ *actual_version_number* ≤ *version_maximum*

    == If version number is present,

    version_minimum = *actual_version_number*

    version_maximum, if defined, is ignored.

    *<>

    Version number *must* be present, and

    version_minimum ≤ *actual_version_number* ≤ *version_maximum*

    *==

    Version number *must* be present, and

version_minimum = *actual_version_number*

version_maximum, if defined, is ignored.

– No version rule, hence always true.

The following values are recognized.

version_value
A string indicating the formats of version numbers which are possible for a file. The legal format type is

s An SCCS (see sccs(1)) version number in a "key version string" of the following form

```
@(#)Copyright Apple Computer, Inc 1986    Version 1.2
```

which is produced by a string containing SCCS keywords as follows:

```
%Z%Copyright Apple Computer, Inc 1986\tVersion %I%
```

where \t is a tab.

version_minimum
A string field containing the earliest allowable version number.

version_maximum
A string field containing the maximum allowable version number. An empty field indicates that there is no maximum allowable version number limit.

12. *checksum_rule*[:*checksum_value*]
A string field containing the checksum rule for interpreting the validity of this file by computing the checksum. The legal rule and its "valid if" condition is

s Compute and compare the checksum, using the algorithm of sum(1), which produces a 16-bit checksum.

– No checksum rule, hence always true.

The recognized value for this rule is

checksum_value
A decimal number containing the checksum value. An empty field, with no numeric value whatsoever, indicates that no checksum is to be computed. (The checksum value of a zero-length file is 00000).

13. *special_rule*
    A string field indicating any special rules and values required for interpreting the validity of this file. (Reserved at this time).
14. *reserved_4*
    Reserved for future use.
15. *reserved_3*
    Reserved for future use.
16. *reserved_2*
    Reserved for future use.
17. *reserved_1*
    Reserved for future use.
18. *description*
    A text field containing a description of the file. The description field *may* be stored separately from the other fields in a special *description* file. In this case, each record in the description file will contain two tab-separated fields: the full pathname of the described file followed by a one line description. The description file, like the rest of the CML will be sorted by filename.

FILES

`/etc/eschatology/init2files`
Those files which are required by autorecovery.

`/etc/eschatology/otherfiles`
The balance of the CML (those files not required for autorecovery).

`/etc/eschatology/descriptions`
The description file.

SEE ALSO

`autorecovery`(8).

NAME

core — format of core image file

DESCRIPTION

The A/UX System writes out a core image of a terminated process when any of various errors occur. See signal(3) for the list of reasons; the most common are memory violations, illegal instructions, bus errors, 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/param.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

setuid(2), signal(3).

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

dialup — modem escape sequence file

DESCRIPTION

/etc/dialup contains one or more entries describing the escape sequences for modems specified by the user (more information to follow). /etc/dialup also contains fields for error strings or error codes returned by modems after a command has been issued. If these fields are not set, the attributes will be set for an Apple modem by default.

The first symbol in an /etc/dialup entry must be an identifier which is taken from mt in remote(4). If an entry is longer than a single line, the lines in the entry must end with a "\". Commands can be one of the following abbreviations, followed by a "=" for a string command or "#" for a numeric command, and then the appropriate command sequence for the particular modem.

ag	repeat the last command	A/
as	attention to signal for modem	AT
at	auto call unit type	generic
cd	return to command mode	;
cr	continuous redial	X2
dp	dial up	D
ec	echo command	E
em	escape command	+++
dm	data mode	O
hu	hang up line	H
vb	verbal command returned from modem	V1

The following are return values from the modem if vb=V1:

ok	the previous command was OK	OK
ct	the modem is connected and is online	CONNECT
nc	the modem has been disconnected	NO CARRIER
er	the previous command is invalid	ERROR

EXAMPLES

If an entry in /etc/remote looked like this:

```
apple:br=1200:at=generic:mt=apple
```

the corresponding entry in /etc/dialup might look like this:

```
apple:as=AT:at=generic:dp=D:cr=X2:\
hu=H:em=+++; ag=A/;ec=E;dm=0:cd=;:ok=OK:\
ct=CONNECT:nc=NO CARRIER: er=ERROR:vb=V1:
```

FILES

/etc/dialup
/etc/remote

SEE ALSO

tip(1C), phones(4), remote(4).

NAME

dir — format of System V directories

SYNOPSIS

```
#include <sys/types.h>
#include <svfs/fsdir.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 inode entry (see fs(4)). The structure of a directory entry as given in the include file is:

```
#ifndef      SVFSDIRSIZ
#define      SVFSDIRSIZ    14
#endif

struct       svfsdirect {
     ino_t   d_ino;
     char    d_name[SVFSDIRSIZ];
};
```

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

fs(4).

NAME

disktab — disk description file

SYNOPSIS

```
#include <disktab.h>
```

DESCRIPTION

disktab is a simple database that describes disk geometries. The format is patterned after the termcap(4) terminal database. Entries in disktab consist of a number of colon-separated fields. The first entry for each disk gives the names that are known for the disk, separated by | characters. The last name given should be a long name fully identifying the disk.

The following list indicates the normal values stored for each disk entry:

Name	**Type**	**Description**
bl	num	Number of blocks per cylinder
ns	num	Number of sectors per track
nt	num	Number of tracks per cylinder
nc	num	Total number of cylinders on the disk
rg	num	Rotational gap

This information is used by the newfs(1M) command.

FILES

/etc/disktab

SEE ALSO

newfs(1M), fs(4).

BUGS

This file shouldn't be necessary. Instead, the information should be stored on each disk.

NAME

dpme — format of disk partition map entries

SYNOPSIS

```
#include <apple/dpme.h>
```

DESCRIPTION

Starting at physical block 1 (offset 512 bytes) of each disk resides a disk partition map. This map describes the layout of the partitions for that disk. The disk partition map consists of one or more disk partition map entries. Each entry corresponds to at most one disk partition. The format of a disk partition map entry is:

```
typedef struct
{
        u16      dpme_signature;
        u16      dpme_reserved_1;
        u32      dpme_map_entries;
        u32      dpme_pblock_start;
        u32      dpme_pblocks;
        DPIDENT  dpme_dpident;
        u32      dpme_lblock_start;
        u32      dpme_lblocks;
        u32      dpme_reserved_2:     23;  /* Bit 9 through 31 */
        u32      dpme_os_specific_1:   1;  /* Bit 8 */
        u32      dpme_os_specific_2:   1;  /* Bit 7 */
        u32      dpme_os_pic_code:     1;  /* Bit 6 */
        u32      dpme_writable:  1;    /* Bit 5 */
        u32      dpme_readable:  1;    /* Bit 4 */
        u32      dpme_bootable:  1;    /* Bit 3 */
        u32      dpme_in_use:    1;    /* Bit 2 */
        u32      dpme_allocated: 1;    /* Bit 1 */
        u32      dpme_valid:     1;    /* Bit 0 */
        u32      dpme_boot_block;
        u32      dpme_boot_bytes;
        u8       *dpme_load_addr;
        u8       *dpme_load_addr_2;
        u8       *dpme_goto_addr;
        u8       *dpme_goto_addr_2;
        u32      dpme_checksum;
        char     dpme_process_id[16];
        u32      dpme_boot_args[32];
        u32      dpme_reserved_3[62];
```

```
} DPME;

#define DPME_SIGNATURE 0x504d  /* Signature value */
#define  DPM_OFF        512    /* byte offset of dp map */
#define  DPISTRLEN      32

struct dpident
{
     char  dpiname[DPISTRLEN];    /* name of partition */
     char  dpitype[DPISTRLEN];    /* type of partition */
};
typedef struct dpident DPIDENT;
```

FIELD DESCRIPTIONS

`dpme_signature`
: This field should always contain the magic number `DPME_SIGNATURE`.

`dpme_reserved_1`
: This field is not used by A/UX.

`dpme_map_entries`
: This field indicates the size of the disk partition map measured in units of disk partition map entries. Since each disk partition map entry is one block big, this field also indicates the number of blocks in the partition map. The value of this field is only meaningful for the first entry in the disk partition map.

`dpme_pblock_start`
: This field indicates the physical block number of the starting block of the physical partition.

`dpme_pblocks`
: This field indicates the number of physical blocks in the partition. This is usually referred to as the size of the physical partition.

`dpme_dpident`
: This field is a structure that contains two string fields. The first field, `dpiname`, contains the name of the partition. The second field, `dpitype`, contains the type of the partition. If the partition name (or type) is less than `DPISTRLEN` bytes long, it must be terminated by a `NULL` (binary zero) byte. An empty partition name or type (first byte `NULL`) is legal. These strings are case sensitive.

`dpme_lblock_start`
: For A/UX partitions, this field will always be zero. This field designates the first data block of the logical partition.

`dpme_lblocks`
: This field designates the number of blocks in the data area of the partition. This is usually referred to as the size of the logical partition. For alternate bad blocking to occur it is necessary for the logical partition to be smaller than the physical partition. Those blocks between the end of the logical partition and the end of the physical partition are usually used for alternate bad blocking.

`dpme_reserved_2`
: This field is not used by A/UX.

`dpme_os_specific_1`
: This field is not used by A/UX.

`dpme_os_specific_2`
: This field is not used by A/UX.

`dpme_os_pic_code`
: This field is not used by A/UX.

`dpme_writable`
: This bit indicates that the creating/controlling operating system allows writing of the logical disk that comprises this partition. Whether or not the writing is allowed by other operating systems and/or processors is not defined. Mainly informative.

`dpme_readable`
: This field is not used by A/UX.

`dpme_bootable`
: This field is not used by A/UX.

`dpme_in_use`
: This field is not used by A/UX.

`dpme_allocated`
: This bit indicates whether or not an operating system has laid claim to the partition described by this entry.

`dpme_valid`
: This bit indicated whether or not this partition entry is valid or not.

dpme_boot_block
This field is not used by A/UX.

dpme_boot_bytes
This field is not used by A/UX.

dpme_load_addr
This field is not used by A/UX.

dpme_load_addr_2
This field is not used by A/UX.

dpme_goto_addr
This field is not used by A/UX.

dpme_goto_addr_2
This field is not used by A/UX.

dpme_checksum
This field is not used by A/UX.

dpme_process_id
This field is not used by A/UX.

dpme_boot_args

dpme_reserved_3
This field is not used by A/UX.

SEE ALSO

dp(1M), pname(1M), altblk(4), bzb(4), gd(7).

FILES

/dev/rdsk/c?d?s31
/usr/include/apple/dpme.h

BUGS

It could be argued that the dpme_boot_args and dpme_signature fields would more appropriately be named dpme_os_specific and dpme_magic, respectively.

NAME

dump.bsd — format of a file system dump

SYNOPSIS

```
#include <sys/types.h>
#include <sys/inode.h>
#include <dumprestor.h>
```

DESCRIPTION

The output of dump.bsd(1M) or the input for restore(1M) contains four distinct items: (1) a header record; (2) two groups of bit map records; (3) a group of records describing directories; and (4) a group of records describing files.

The format of the header record and of the first record of each description is given in the include file <dumprestor.h>.

```
#define NTREC           20
#define MLEN            16
#define MSIZ            4096

#define TS_TAPE         1
#define TS_INODE        2
#define TS_BITS         3
#define TS_ADDR         4
#define TS_END          5
#define TS_CLRI         6
#define MAGIC           (int)60011
#define CHECKSUM        (int)84446
struct  spcl
{
        int     c_type;
        time_t  c_date;
        time_t  c_ddate;
        int     c_volume;
        daddr_t c_tapea;
        ino_t   c_inumber;
        int     c_magic;
        int     c_checksum;
        struct  dinode  c_dinode;
        int     c_count;
        char    c_addr[BSIZE];
} spcl;

struct  idates
{
        char    id_name[16];
        char    id_incno;
        time_t  id_ddate;
};
```

NTREC is the number of 1024 byte records in a physical block for the backup device. MLEN is the number of bits in a bit map word. MSIZ is the number of bit map words.

The TS_ entries are used in the c_type field to indicate what sort of header it is. The types and their meanings are as follows:

TS_TAPE	Volume label.
TS_INODE	A file or directory follows. The c_dinode field is a copy of the disk inode and contains bits telling what sort of file it is.
TS_BITS	A bit map follows. This bit map has a 1 bit for each inode that was dumped.
TS_ADDR	A subrecord of a file description. (See c_addr later.)
TS_END	End of media record.
TS_CLRI	A bit map follows. This bit map contains a 0 bit for all inodes that were empty on the file system when dumped.
MAGIC	All header records have this number in c_magic.
CHECKSUM	Header records checksum to this value.

The fields of the header structure are as follows:

c_type	The type of the header.
c_date	The date the dump was taken.
c_ddate	The date the file system was dumped.
c_volume	The current volume number of the dump.
c_tapea	The current number of this (1024-byte) record.
c_inumber	The number of the inode being dumped if of type TS_INODE.
c_magic	This contains the value MAGIC above, truncated as needed.
c_checksum	This contains whatever value is needed to make the record sum to CHECKSUM.
c_dinode	This is a copy of the inode as it appears on the file system (see fs(5)).
c_count	The count of characters in c_addr.
c_addr	An array of characters describing the blocks of the dumped file. A character is zero if the block associated with that character was not present on the file system, otherwise the character is nonzero. If the block was not present on the file system, no block was dumped; the block will be restored as a hole in the file. If there is not sufficient space in

this record to describe all of the blocks in a file, TS_ADDR records will be scattered through the file, each picking up where the last left off.

Each volume, except the last, ends with a tapemark (read as an end of file). The last volume ends with a TS_END record and then the tapemark.

The structure idates describes an entry in the file /etc/dumpdates where dump history is kept. The fields of the structure are

id_name	The name of dumped file system, /dev/id_name.
id_incno	The level number of the dump media (see dump.bsd(1M)).
id_ddate	The date of the incremental dump in system format (see types(5)).

FILES

/etc/dumpdates

SEE ALSO

dump.bsd(1M), restore(1M), fs(4), types(5).

NAME

errfile — error-log file format

DESCRIPTION

When hardware errors are detected by the system, 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 */
};
```

The permissible record types are as follows:

```
#define E_GOTS     010   /* start for the UNIX/TS */
#define E_GORT     011   /* start for the UNIX/RT */
#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 */
```

Some records in the error file are of an administrative nature. These include the startup record that is entered into the file when logging is activated, the stop record that is written if the daemon is terminated "gracefully", and the time-change record that is 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 */
};
#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 {
       uint        e_saddr;  /* stray loc or device addr */
};
```

Generation of memory subsystem errors is not supported in this release.

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_cyloff;   /* logical dev start cyl */
    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 */
};
```

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_FORMAT  010  /* Formatting Disk*/
#define E_ERROR   020  /* I/O failed */
```

SEE ALSO

errdemon(1M).

NAME

ethers — Ethernet address to hostname database or YP domain

DESCRIPTION

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

ethernet-address hostname

Items are separated by any number of blanks and/or tabs. Use # to introduce a single line or midline comment.

The standard form for *ethernet-address* is *x*:*x*:*x*:*x*:*x*:*x*: where *x* is a hexadecimal number between 0 and 255, representing one byte. The address bytes are always in network order. *hostname* may contain any printable character other than a space, tab, newline, or comment character. The hostnames in the ethers file should correspond to the hostnames in the /etc/hosts file (see hosts(4)).

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

FILES

/etc/ethers

SEE ALSO

ethers(3N), hosts(4).

NAME

exports — NFS file systems being exported

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 netgroup(4) or 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/netgroup and then 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.

EXAMPLES

```
/usr        clients                # export to my clients
/usr/local                         # export to the world
/usr        phoenix sun sundae     # export to only these
                                     machines
```

FILES

/etc/exports

SEE ALSO

mountd(1M), netgroup(4).

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 in 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. See `aouthdr`(4) for the structure of the optional `a.out` header. The valid magic number is

```
#define   MC68MAGIC   0520    /* magic number */
```

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

```
#define F_RELFLG  00001 /* relocation entries stripped */
#define F_EXEC    00002 /* file is executable */
#define F_LNNO    00004 /* line numbers stripped */
#define F_LSYMS   00010 /* local symbols stripped */
#define F_MINMAL  00020 /* minimal object file */
#define F_UPDATE  00040 /* update file, ogen produced */
#define F_SWABD   00100 /* file is "pre-swabbed" */
#define F_AR16WR  00200 /* 16-bit DEC host */
#define F_AR32WR  00400 /* 32-bit DEC host */
#define F_AR32W   01000 /* non-DEC host */
#define F_PATCH   02000 /* "patch" list in opt hdr */
#define F_NODF    02000 /* "patch" list in opt hdr */
```

SEE ALSO

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

NAME

finstallrc — finstall default configuration file

SYNOPSIS

/etc/finstallrc

DESCRIPTION

You can use the .finstallrc and /etc/finstallrc files to specify the default options used with finstall, such as whether finstall should prompt for which floppy drive to use. The variables that can be set for finstall are as follows:

CTL_ASKDRIVE
: determines if finstall should prompt for which floppy drive to use.

CTL_ASKINSTALL
: determines if finstall should prompt for the directory to install the software under.

CTL_CHECKSPACE
: determines if finstall should check for enough space to install the software.

CON_TRIES
: specifies the number of times allotted to attempt to answer a prompt.

CTL_ALLOWRC
: determines whether the .finstallrc file should be used.

CTL_TAKEDEFAULT
: determines if finstall should use default answers.

The default values for these variables are as follows:

```
CTL_ASKDRIVE        =1
CTL_ASKINSTALL      =1
CTL_CHECKSPACE      =1
CON_TRIES           =5
CTL_ALLOWRC         =1
CTL_TAKEDEFAULT     =0
```

You can change the value of the default variables with results described as follows:

CTL_ASKDRIVE
: != 0: instructs finstall to prompt for which floppy drive to use for installation.

== 0: instructs `finstall` to use the right-hand floppy drive for installation.

`CTL_ASKINSTALL`
: != 0: instructs `finstall` to prompt for the installation directory.
== 0: instructs `finstall` to use the directory specified by the software developer as the default installation directory. If the software developer did not specify a directory, `finstall` uses the current working directory as the installation directory.

`CTL_CHECKSPACE`
: != 0: instructs `finstall` to check for enough space on the installation directory to install the software.
== 0: instructs `finstall` to proceed with installation without checking for available space.

`CON_TRIES`
: ==*n*: *n* specifies the number of times allotted to attempt to answer a prompt.

`CTL_ALLOWRC`
: != 0: instructs `finstall` to not use a `.finstallrc` file in the current working directory.
== 0: instructs `finstall` to use a `.finstallrc` file in the current working directory.

`CTL_TAKEDEFAULT`
: != 0: instructs `finstall` to print the prompt on the screen but to use the default answer rather than waiting for a user response.
== 0: instructs `finstall` to print the prompt on the screen and wait for a response from the user.

FILES

`/etc/finstallrc`
`.finstallrc`

SEE ALSO

`finstall`(1M).

NAME

fs — file systems

DESCRIPTION

A/UX® supports System V file systems (SVFS) and Berkeley 4.2 file systems (UFS). (See svfs(4) and ufs(4) for details about file-system organization.) A/UX does not support Macintosh® file systems as mountable file systems. However, the A/UX finder may read and write these file systems. Please see *Inside Macintosh, Volume II* for a description of the original Macintosh file system and *Inside Macintosh, Volume IV* for a description of the hierarchical file system (HFS).

mkfs is used to create SVFS file sytems.

newfs is used to create UFS file systems. tunefs can be used to change the dynamic parameters of a UFS.

SEE ALSO

mkfs(1M), newfs(1M), tunefs(1M), svfs(4), ufs(4).

NAME

`fspec` — syntax for format lines for `newform`

DESCRIPTION

It is sometimes convenient to maintain text files on the A/UX system with nonstandard tabs, (i.e., 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 A/UX system commands. A format specification occurring in the first line of a text file specifies how tabs are to be expanded in the remainder of the file.

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

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

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 prefixed.

`m`*margin* The `m` parameter specifies a number of spaces to be prefixed 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 — static information about file systems

SYNOPSIS

`#include <mntent.h>`

DESCRIPTION

The file /etc/fstab describes the file systems and swapping partitions used by the local machine. It can be modified with a text editor by the system administrator. The file is read by commands that mount, unmount, and check the consistency of file systems; it is also read by the system in providing swap space. Because there is an appropriate mount request in the /etc/rc startup file, any file systems described in /etc/fstab (other than those of type ignore or with mount option noauto) are mounted automatically whenever multi-user mode is entered.

The /etc/fstab file consists of a number of lines in the following format

fsname dir type opts freq passno

For example

```
/dev/xy0a / 5.2 rw,noquota 1 2
```

The field *freq* is optionally used by dump.bsd(1M) to help report which file systems need to be dumped. *passno* is used by fsck(1M) to help select which file systems to check. For example, fsck -p2 checks all the 5.2 file systems listed in /etc/fstab with *passno* greater than or equal to 2.

The entries from this file are accessed using the routines in getmntent(3), which returns a structure of the following form:

```
struct mntent {
    char    *mnt_fsname; /* file system name */
    char    *mnt_dir;    /* file system path prefix */
    char    *mnt_type;   /* 4.2, 5.2, nfs, swap,
                            or ignore */
    char    *mnt_opts;   /* rw, ro, noquota, quota, noauto,
                            hard, soft */
    int     mnt_freq;    /* dump frequency, in days */
    int     mnt_passno;  /* pass # on parallel fsck */
};
```

Fields are separated by white space; a # as the first nonwhite character indicates a comment.

The mnt_type field determines how the mnt_fsname and mnt_opts fields will be interpreted. Here is a list of the file system types currently supported, and the way each of them interprets these fields.

4.2/5.2

mnt_fsname	Must be a block device.
mnt_opts	Valid options are ro, rw, quota, noquota, noauto.

NFS

mnt_fsname	The path on the server of the directory to be served.
mnt_opts	Valid options are ro, rw, quota, noquota, noauto, hard, soft.

SWAP

mnt_fsname	Must be a block device swap partition.
mnt_opts	Ignored.

If the mnt_opts field contains noauto, the entry will be ignored during a mount -a command, allowing definition of fstab entries for commonly used file systems not mounted automatically.

If the mnt_type is specified as ignore then the entry is ignored. This is useful to show disk partitions not currently used.

The /etc/fstab file is only read by programs and never written by them; it is the duty of the system administrator to maintain this file. The order of records in /etc/fstab is important because fsck, mount, and umount process the file sequentially; file systems must appear following the file systems they are mounted in.

Note that listing a file system as type swap will not cause the system to mount the file system as a swap area; to do that, you must use the swap command.

FILES

/etc/fstab

SEE ALSO

dump.bsd(4), fsck(1M), mount(1M), swap(1M), getmntent(3).

NAME

fstypes — name-mapping information for file systems

SYNOPSIS

```
#include <sys/fstypent.h>
```

DESCRIPTION

/etc/fstypes contains information about file-system types. It can be modified by the system administrator using a text editor. The file is used by commands that need to know the type of a specified file system. It is also used by commands to determine the location of file-system-dependent utilities.

The /etc/fstypes file consists of lines in the following format:

numeric-type name-list [*pathname-list*]

For example:

```
0      5.2,svfs,s5 /etc/fs/5.2:/etc/fs/svfs
```

The fields are separated by white space; a # as the first character indicates a comment. A # after *name-list* or *path-list* indicates that the rest of the line is a comment.

The placeholder *numeric-type* is the integer type that is passed to fsmount. See fsmount(2). These are defined in <sys/mount.h>. The *name-list* is a comma-separated list of character strings that describe the file-system type. At least one of these is defined in <mntent.h>. The *pathname-list* is a colon-separated list of pathnames. These pathnames indicate where utility programs associated with the file-system type reside. If this field is empty, the default location is /etc/fs/name-list.

The entries in this file are accessed using fstypent, which reads the next entry from the file and returns a pointer to a struct fstypent. This structure is defined in <sys/fstypent>as:

```
struct fstypent {
       int fstype;
       char **typelist;
       char *pathlist;
};
```

FILES

/etc/fstypes

SEE ALSO

getmntent(3), fstypent(3), fs(4), fstab(4).

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` prompt should look like. It also supplies the speed to try next if the user indicates the current speed is not correct by typing an interrupt character.

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

label# initial-flags # final-flags # flow-control # 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 that `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 flags `SANE` or `SANE2` take 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. Flag attributes are added from left to right, flags that start with a ~ are subtracted, e.g., SANE ~PARENB. This field specifies what type of flow control to use on the line. The currently allowed settings are APPLE (for apple flow control), DTR (for DTR flow control), MODEM (for modem control), and FLOW (for hardware flow control). These modes can also be turned off by using the ~ as a prefix.

login-prompt This entire field is printed as the *login-prompt*. Unlike the above fields where white space is ignored (a space, tab or *newline*), 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 are linked together in this fashion, into a closed set; For instance, 2400 linked to 1200, which in turn is linked to 300, which finally is linked to 2400.

If getty is called without a second argument, then the first entry of /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 300 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.

The following four symbols define the SANE state.

```
# define ISANE (BRKINT | IGNPAR | ISTRIP | ICRNL | IXON)
# define OSANE (OPOST  | ONLCR)
# define CSANE (CS7 | PARENB | CREAD)
# define LSANE (ISIG | ICANON | ECHO | ECHOK)
```

FILES

/etc/gettydefs

SEE ALSO

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

NAME

`group` — group file

SYNOPSIS

`/etc/group`

DESCRIPTION

`group` contains for each group the following information:

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

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

This file resides in the `/etc` directory. 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.

A group file can have a line beginning with a plus (+), which means to incorporate entries from the yellow pages. There are two styles of + entries: All by itself, + means to insert the entire contents of the yellow pages group file at that point; +*name* means to insert the entry (if any) for *name* from the yellow pages at that point. If a + entry has a nonnull password or group member field, the contents of that field will overide what is contained in the yellow pages. The numerical group ID field cannot be overridden.

EXAMPLES

```
+myproject:::carolyn, jennifer
+:
```

If these entries appear at the end of a group file, then the group `myproject` will have members `carolyn` and `jennifer`, and the password and group ID of the yellow pages entry for the group `myproject`. All the groups listed in the yellow pages will be pulled in and placed after the entry for `myproject`.

FILES

`/etc/group`
`/etc/yp/group`

SEE ALSO

passwd(1), setgroups(2), crypt(3), initgroups(3), passwd(4).

BUGS

The passwd(1) command won't change group passwords.

NAME

HOSTNAME — hostname and domainname database

DESCRIPTION

HOSTNAME resides in the /etc directory and consists of one line containing the following items of information

hostname domainname

Items are separated by any number of blanks and/or tabs. There must be no white space at the beginning of the line.

hostname is the name of the local host machine and *domainname* is the name of the Yellow Pages domain on which the local host resides.

EXAMPLES

```
magic        apple
```

FILES

/etc/HOSTNAME

SEE ALSO

hostname(1), domainname(1), chgnod(1M).
A/UX Installation Guide
RFC-882, RFC-883, RFC-920, RFC-921, RFC-952, RFC-953, RFC-973, RFC-974 (DNN Network Information Center, SRI International)

NAME

hosts — host name database

DESCRIPTION

The hosts file contains information regarding the known hosts on the DARPA Internet. For each host a single line should be present with the following information:

official host name
Internet address
aliases

Items are separated by any number of blanks or tab characters. A # indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file. This file is normally created from the official host data base maintained at the Network Information Control Center (NIC), though local changes may be required to bring it up to date regarding unofficial aliases and unknown hosts.

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

FILES

/etc/hosts

NAME

hosts.equiv — list of trusted hosts

DESCRIPTION

hosts.equiv resides in directory /etc and contains a list of trusted hosts. When an rlogin(1) or remsh(1) request from such a host is made, and the initiator of the request is in /etc/passwd, then no further validity checking is done. That is, rlogin does not prompt for a password, and remsh completes successfully. So a remote user is "equivalenced" to a local user with the same user ID when the remote user is in hosts.equiv.

The format of hosts.equiv is a list of names, as in this example:

```
host1
host2
+@group1
-@group2
```

A line consisting of a simple host name means that anyone logging in from that host is trusted. A line consisting of +@group means that all members of that network group are trusted. A line consisting of −@group means that members of that group are not trusted. Programs scan hosts.equiv linearly, and stop at the first hit (either positive for hostname and +@ entries, or negative for −@ entries). A line consisting of a single + means that everyone is trusted.

The .rhosts file has the same format as hosts.equiv. When user *x* executes rlogin or remsh, the .rhosts file from *X*'s home directory is conceptually concatenated onto the end of hosts.equiv for permission checking. However, −@ entries are not sticky. If a user is excluded by a minus entry from hosts.equiv but included in .rhosts, then that user is considered trusted. In the special case when the user is root, then only the /.rhosts file is checked.

It is also possible to have two entries (separated by a single space) on a line of these files. In this case, if the remote user is equivalenced by the first entry, then that user is allowed to log in as any member of the second entry. Thus

```
sundown john
```

allows anyone from `sundown` to log in as `john`, and

```
+@group1 +@group2
```

allows any member of *netgroup1* to log in as a member of *netgroup2*.

FILES

`/etc/hosts.equiv`

SEE ALSO

rlogin(1), remsh(1), netgroup(4).

NAME

inittab — script for the init process

DESCRIPTION

The inittab file supplies the script for the role init plays as a general process dispatcher. The process that constitutes the majority of the process dispatching activities of init 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 may be inserted in the *process* field using the sh(1) convention for comments. Comments for lines that spawn getty processes are displayed by the who(1) command. It is expected that they will 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 are

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

rstate This defines the *run level* in which this entry is to be processed. The entry, *run levels* effectively corresponds to a configuration of processes in the system. That is, each process spawned by init is assigned a *run level* or *run levels* in which it is allowed to exist. The *run levels* are represented by a number ranging from 0 through 6. As an example, if the system is in *run level* 1, only those entries having a 1 in the *rstate* field will be processed. When init is requested to change *run levels*, all processes which do not have an entry in the *rstate* field for the target *run level* will be 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 0–6. If no *run level* is specified, then the process is assumed to be valid at all *run levels* 0–6.

There are three other values, `a`, `b`, and `c`, which can appear in the *rstate* field, even though they are not true *run levels*. Entries which have these characters in the *rstate* field are processed only when the `init` (see `init`(1M)) process requests them to be run (regardless of the current *run level* of the system). They differ from *run levels* in that `init` can never enter *run level* `a`, `b`, or `c`. Also, a request for the execution of any of these processes does not change the current *run level*. Furthermore, a process started by an `a`, `b`, or `c` command is not killed when `init` changes levels. They are only killed 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, then start the process (do not wait for its termination, that is, continue scanning the `inittab` file), and when it dies restart the process. If the process currently exists, then do nothing and continue scanning the `inittab` file.

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

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

`boot` — The entry is to be processed only at the boot-time read of the `inittab` file. `init` is to start the process, not wait for its termination; and when it dies, 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 only at the boot-time read by `init` of the `inittab` file. `init` is to start the process, wait for its termination and, when it dies, not restart the process.

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

`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 via 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 only scanned when `init` is initially invoked. `init` uses this entry, if it exists, to determine which *run level* to enter initially. It does this by taking the highest *run level* specified in the `rstate` field and using that as its initial state. If the *rstate* field is empty, this is interpreted as `0123456` and so `init` will enter *run level* `6`. Also, the `initdefault` entry can use `s` to specify that `init` start in the *SINGLE USER* state. Additionally, if `init` does not find an `initdefault` entry in `/etc/inittab`, then it will request 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 This is 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

`sh`(1), `who`(1), `getty`(1M), `exec`(2), `open`(2), `signal`(3).

NAME
inode — format of a System V inode

SYNOPSIS
```
#include <sys/types.h>
#include <svfs/inode.h>
```

DESCRIPTION
An inode for a plain file or directory in a file system has the following structure defined by `<svfs/inode.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[40]; /* disk block addresses */
#define di_gen; di_addr[39]
        time_t  di_atime;     /* time last accessed */
        time_t  di_mtime;     /* time last modified */
        time_t  di_ctime;     /* time created */
};

/*
* the 40 address bytes:
*       39 used; 13 addresses
*       of 3 bytes each.
*/
```

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

FILES
/usr/include/svfs/inode.h

SEE ALSO
`stat`(2), `fs`(4), `types`(5).

NAME

ioctl.syscon — console terminal settings file

SYNOPSIS

/etc/ioctl.syscon

DESCRIPTION

The file /etc/ioctl.syscon contains information about the ioctl states of the A/UX virtual terminal console. This file is created by init(1M) when the system is put into single-user mode, and it is read by the init process when init first comes up.

The information contained in /etc/ioctl.syscon is used to set the terminal modes on the initial console emulator. It is used primarily to preserve reasonable values for terminal settings across system reboots (instead of using the driver-imposed defaults).

The ioctl.syscon file consists of 16 colon-separated fields, closely resembling the output of the command

```
stty -g
```

For example, a sample /etc/ioctl.syscon file looks like this:

```
526:5:bd:3b:0:3:1c:7f:15:4:0:0:0:0:0:0
```

while the stty -g command on the console terminal would produce the following output:

```
526:5:bd:3b:3:1c:7f:15:4:0:0:0
```

The primary difference is that the ioctl.syscon file contains four additional fields corresponding to the termcb structure, an undocumented artifact of System III. These four fields are always zero. The remaining fields correspond to the fields of the termio structure; for an explanation of these fields, see termio(7).

If the /etc/ioctl.syscon file becomes damaged, the system may refuse to accept input from the console terminal during the boot process. To remedy this situation, it is safest simply to remove the file altogether from within the A/UX Startup shell environment, allowing the default settings to be established once again. The driver defaults are reasonable and will allow the system to boot successfully. A corrected version of the file will then be generated when the system is booted into multi-user mode. See StartupShell(8) for details on performing A/UX file system operations from the A/UX Startup shell.

FILES

/etc/ioctl.syscon

SEE ALSO

stty(1), init(1M), termio(7), StartupShell(8).

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

iwmap — format of iwprep(1) character map description files

SYNOPSIS

/usr/lib/font/*device*/MAP.*

DESCRIPTION

A *map file* specifies a character code for a troff character name. A complete list of the troff character names may be found in the "nroff/troff Reference" in *A/UX Text Processing Tools.*

Each map file line has the synopsis:

code charname...

where *code* and *charname* are described as follows:

code Any valid C eight-bit integer constant, including decimal, octal, and hexadecimal forms.

charname A one or two character name. One character names are used to specify standard ASCII characters (e.g., a, b, c, 1, 2, 3). Two character names are used to specify special characters. There are two forms of the special character names in troff input. The first is a simple two character name (e.g., \-, \|). The second is a four character name (e.g., \(*A, \(dg). For the two character name of special characters, you specify the entire name (i.e., \- for \-). For the four character name of special characters, you specify just the last two characters (i.e., dg for \(dg).

EXAMPLES

An example map file for specifying the code for a dash, hyphen, and long dash to the same character is:

```
055      - hy -
```

Examine the map files in /usr/lib/font/deviw for further examples.

FILES

/usr/lib/font/deviw

SEE ALSO

iwprep(1).

NAME

linenum — line number entries in a common object file

SYNOPSIS

```
#include <linenum.h>
```

DESCRIPTION

The C compiler 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 reference 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

magic — magic number file for file command

DESCRIPTION

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

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

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

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

byte	A one-byte value.
short	A two-byte value.
long	A four-byte value.
string	A string of bytes.

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

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

Numeric values may be preceded by a character indicating the operation to be performed. The character may be an =, to specify that the value from the file must equal the specified value, a <, to specify that the value from the file must be less than the specified value, a >, to specify that the value from the file must be greater than the specified value, or an `x` to specify that any value will match. If the character is omitted, it is assumed to be =.

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

message The message to be printed if the comparison succeeds. If the string contains a `printf`(3S) format specification, the value from the file (with any specified masking performed) is printed using the message as the format string.

Some file formats contain additional information which is to be printed along with the file type. A line which begins with the character > indicates additional that tests and messages are to be printed. If the test on the line preceding the first line with a > succeeds, the tests specified in all the subsequent lines beginning with > are performed, and the messages are printed if the tests succeed. The next line which does not begin with a > terminates this command.

FILES

`/etc/magic`

SEE ALSO

`file`(1).

BUGS

There should be more than one level of subtests, with the level indicated by the number of > at the beginning of the line.

NAME

`master` — master kernel configuration files

DESCRIPTION

Master files are used by `autoconfig`(1M) to obtain device information that is necessary to configure new kernels. Master files are located in `/etc/master.d`.

Master files can contain up to three order-dependent lines of information: a device identifier, a dependency statement, and a device specification. The device-identifier and dependency-statement lines are optional and precede the device specification, as shown below:

device-identifier
dependency-statement
device-specification

Device Identifier

The device identifier provides optional information that is useful only for slot device drivers. Each slot card stores a board ID number and a version number in its ROM. The device identifier is used to specify a particular slot card and, optionally, a range of version numbers, as shown below:

`id` *board-id serial*

where *board-id* is an integer value that matches the board ID that is stored in a slot card's ROM. For example, *board-id* with a value of 8 indicates the EtherTalk™ card. The placeholder *serial* is an optional number or number range. If present, *serial* is compared with the slot card's version number. If the comparison fails, `autoconfig` terminates. The placeholder *serial* can be specified as:

number
The slot card's version number must match *number*.

number–
The slot card's version number must be less than or equal to *number*.

–number
The slot card's version number must be greater than or equal to *number*.

number1–number2
The slot card's version number must be within the range

specified by *number1–number2*.

If *serial* is not specified, `autoconfig` does not check the slot card's version number.

Dependency Statements

Dependency statements can be used to specify modules that must be included or excluded in the resulting kernel for proper operation of the subject driver. Dependency statements can have several forms, from simple to complex:

verb namelist
`if` *filename verb namelist*
`if` *expression verb namelist*

The possible values for *verb*, *namelist*, *filename*, and *expression* are described below:

verb
: The keyword `include` or `exclude`. `include` tells `autoconfig` to include the modules specified in *namelist* in the resulting kernel. `exclude` tells `autoconfig` to exclude the modules specified in *namelist* from the resulting kernel.

namelist
: A comma-separated list of module names.

filename
: The name of another master file in the current directory or a period (`.`), which indicates the current master file. If *filename* exists, the modules specified in *namelist* are included in the resulting kernel. If *filename* does not exist, the modules specified in *namelist* are excluded.

expression
: An *expression* constructed from filenames and operators. If the evaluation of the expression is TRUE, the modules specified in *namelist* are included in the resulting kernel. If the evaluation of the expression is FALSE, the modules specified in *namelist* are excluded. The following operators, listed from highest to lowest priority, can be used to construct *expression*:

!	NOT
&	AND
|	OR

Parentheses can be used to override the default priority. The following examples use parentheses to demonstrate the default priority of the operators:

`a | b & c` is equivalent to `a | (b & c)`
`!a & b` is equivalent to `(!a) & b`

Device Specification

The device specification provides information that `autoconfig` must know to produce a complete and working kernel. A device specification is comprised of the following six fields:

flags
vectors
prefix
major-number
maximum-devices
interrupt-level

The fields must appear on a single line in the master file in the order shown above and must be separated by one or more blanks or tabs. Each field is described below:

flags
One or more of the following characters:

`a` Tell `autoconfig` to create *prefix*`cnt` and *prefix*`addr` data structures for this module. The value of *prefix* is discussed below.

`b` Tell `autoconfig` to create a `bdev` switch entry for this module.

`c` Tell `autoconfig` to create a `cdev` switch entry for this module.

`l` Tell `autoconfig` to create a line discipline switch entry for this module.

`m` Tell `autoconfig` to create a Streams entry for this module.

`n` Tell `autoconfig` that this module uses a network interface (TCP/IP).

`p`*opt*
Tell `autoconfig` that this module has an initialization routine. `autoconfig` generates code that calls the initialization routine at the point in the dur-

ing system boot specified by *opt*, which can be any of the following characters:

- `f` Call this module's initialization routine first, before any other initialization occurs. Interrupts are disabled.
- `s` Call this module's initialization routine after any `pf` modules. Interrupts are disabled.
- `n` Call this module's initialization routine after any `pf` and `ps` modules but prior to enabling interrupts. If `p`*opt* is not specified, `n` is the default.
- `0` Call this module's initialization routine after enabling interrupts.
- `1` Call this module's initialization routine before entering `/etc/init`.

`s` Tell `autoconfig` that this module is a software module that does not drive a hardware device. Of the other possible values for *flags*, only the `p` flag can be used with the `s` flag.

`t` Tell `autoconfig` that this module is a character device driver that requires a `tty` structure. The `t` flag must be used with the `c` flag.

`v`*opt*
Tell `autoconfig` to link this driver to the interrupt vector mechanism. Currently, the only valid value of *opt* is `s`, which tells the kernel to decode slot-based interrupts and call the interrupt routine of this driver when the card generates an interrupt.

`x` Tell `autoconfig` that this module is a Streams module. Only the `p` flag can be used with the `x` flag.

`S`*opt*
Specify *opt* as one of the following characters:

- `e` Tell `autoconfig` that this module contains a special *exit* routine.

f
: Tell `autoconfig` that this module contains a special *fork* routine.

x
: Tell `autoconfig` that this module contains a special *exec* routine.

vectors
The number of interrupt vectors that a particular controller can generate. For hardware device drivers, this value must be a nonzero integer. For drivers that receive slot interrupts, this number is 1 because each controller can generate only one interrupt. For software modules, that do not drive a hardware device this value should be a hyphen (–).

prefix
The prefix used in the driver's open, close, read, write, ioctl, print, select, and strategy routines. For example, if the driver's open routine is called `bddopen`, the prefix is `bdd`. The placeholder *prefix* must be between three and eight characters long. Valid characters are alphanumerics and the underline (_) character. To maintain consistency, *prefix* should also be the name of the master file.

major-number
The value that is assigned as the major number for the device driver. This value should always be a hyphen (–). When a hyphen is specified in this field, `autoconfig` assigns the first available major number to the device. Letting `autoconfig` assign the major number guarantees a unique major number for each device driver and prevents conflict between two or more device drivers.

maximum-devices
Either a hyphen (–) for software modules or a nonzero integer for hardware device drivers. The integer value is the number of devices the controller supports.

interrupt-level
The highest-priority interrupt level used by the controller. For software modules, this value should be a hyphen (–). For slot-based devices, all of which interrupt at `spl1`, this value should be 1.

EXAMPLES

The following master file is for a block device driver:

```
if . include SCSI
bca    -      bdd    -      2      1
```

The `if . include SCSI` statement forces the inclusion of another module, SCSI (the SCSI Manager), on which this device depends. The `b` and `c` flags indicate that the driver is used as both a block and a character device driver, so `autoconfig` will create entries for this device in both the `bdevsw` and `cdevsw` tables. The `a` flag tells `autoconfig` to create the `bddcnt` and `bddaddr` data structures.

Because this device receives interrupts via the SCSI Manager, the hyphen (–) in the second field is used to tell `autoconfig` that this device does not receive interrupts directly. The device's prefix is `bdd`, and, because the fourth field contains a hyphen, `autoconfig` assigns the device driver's major number.

The `2` in the fifth field indicates that there are two devices per controller, and the `1` in the sixth field indicates that the device's interrupt level is `spl1`.

FILES

`/etc/master.d` Default location of master files

SEE ALSO

`autoconfig`(1M).
Building A/UX Device Drivers, which is available from APDA™.

NAME

mtab — mounted file system table

DESCRIPTION

mtab resides in directory /etc and contains a record of all file systems mounted on this machine. Whenever a mount is done, an entry is made in the mtab file. umount removes entries. The table is a series of lines with form identical to that of /etc/fstab.

FILES

/etc/mtab

SEE ALSO

mount(1M), shutdown(1M), umount(1M), fstab(4).

NAME

NETADDRS — network address database

DESCRIPTION

The NETADDRS file resides in /etc and contains information regarding the network addresses of each EtherTalk board on the local machine. For each board, a single line should be present with the following items of information:

unit-number internet-address broadcast-address netmask

Items are separated by any number of blanks and/or tab characters. Lines must not begin with blanks or tabs. *netmask* should be blank if subnets are not being supported.

EXAMPLES

The following is a sample NETADDRS file for a machine on two networks; only the second is subnetted.

```
0   89.53       89.0
1   91.1.0.48   91.1.0.0   255.255.0.0
```

FILES

/etc/NETADDRS

SEE ALSO

autoconfig(1M), ifconfig(1M).
A/UX Network System Administration
RFC-917, RFC-922, RFC-944, RFC-950 (DDN Network Information Center , SRI International)

NAME

`netgroup` — list of network groups

DESCRIPTION

`netgroup` defines network-wide groups, which are used for permission checking when doing remote mounts, remote logins, and remote shells. Each line of the `netgroup` file defines a group and has the format

groupname member1 member2 ...

where *member1* is either another group name or a triple of the form

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

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

`universal (,,)`

defines a group to which everyone belongs.

Network groups are accessed through the yellow pages. The database actually used by the yellow pages are in the two files

/etc/yp/*domainname*/netg.dir
/etc/yp/*domainname*/netg.pag

These files can be created from `/etc/netgroup` using `makedbm`(1M).

FILES

/etc/netgroup
/etc/yp/*domainname*/netg.dir
/etc/yp/*domainname*/netp.pag

SEE ALSO

`makedbm`(1M), `ypserv`(1M), `getnetgrent`(3), `exports`(4).

NAME

networks — network name database

DESCRIPTION

The networks file contains information regarding the known networks which comprise the DARPA Internet. For each network a single line should be present with the following information:

official network name
network number
aliases

Items are separated by any number of blanks and/or tab characters. A "#" indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file. This file is normally created from the official network data base maintained at the Network Information Control Center (NIC), though local changes may be required to bring it up to date regarding unofficial aliases and/or unknown networks.

Network number may be specified in the conventional "." notation using the inet_network() routine from the Internet address manipulation library, inet(3N). Network names may contain any printable character other than a field delimiter, newline, or comment character.

FILES

/etc/networks

SEE ALSO

getnetent(3N).

BUGS

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

NAME

`passwd` — password file

SYNOPSIS

`/etc/passwd`

DESCRIPTION

The `passwd` file contains for each user the following information:

name User's login name; contains no uppercase characters and must not be greater than eight characters long.

password encrypted password as well as aging information

numeric-user-ID
This is the user's ID in the system and it must be unique.

numeric-group-ID
This is the number of the group that the user belongs to.

real-name In some versions of UNIX, this field also contains the user's office, extension, home phone, and so on. For historical reasons this field is called the GCOS field.

default-working-directory
The directory that the user is positioned in when they log in — this is known as the 'home' directory.

shell program to use as Shell when the user logs in.

The user's real name field may contain "&", meaning insert the login name.

The password file is an ASCII file. Each field within each user's entry is separated from the next by a colon. Each user 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.

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 numeric user ID to names.

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 effected for a particular user if his encrypted password in the password file 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 superuser.)

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 login after his password has expired will be forced to supply a new one. The next character, *m* say, denotes the minimum period in weeks which must expire before the password may 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 (i.e., / = 1 week; z = 63 weeks). If *m* = *M* = 0 (derived from the string `.` or `..`) the user will be forced to change his password the next time he logs in (and the "age" will disappear from his entry in the password file). If *m* > *M* (signified, e.g., by the string `./`) only the superuser will be able to change the password.

The `passwd` file can also have line beginning with a plus (+), which means to incorporate entries from the yellow pages. There are three styles of + entries: all by itself, + means to insert the entire contents of the yellow pages password file at that point; +*name* means to insert the entry (if any) for *name* from the yellow pages at that point; +@*name* means to insert the entries for all members of the network group *name* at that point. If a + entry has a nonnull password, directory, GCOS, or shell field, they will overide what is contained in the yellow pages. The numeric user ID and group ID fields cannot be overridden.

EXAMPLES

Here is a sample `/etc/passwd` file:

```
root:q.mJzTnu8icF.:0:10:God:/:/bin/csh
ja:6k/7KCFRPNVXg:508:10:Jerry Asher:/usr2/ja:/bin/csh
+melissa:
+@documentation:no-login:
+:::Guest
```

In this example, there are specific entries for users `root` and `ja`, in case the yellow pages are out of order. The user `melissa` will have her password entry in the yellow pages incorporated without change; anyone in the netgroup `documentation` will have their password field disabled, and anyone else will be able to log in with their usual password, shell, and home directory, but with a GCOS field of `Guest`.

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

FILES

`/etc/passwd`

SEE ALSO

`login`(1), `passwd`(1), `vipw`(1M), `crypt`(3), `getpwent`(3), `group`(4).

NAME

phones — remote host telephone number database

DESCRIPTION

The file /etc/phones contains the system-wide private telephone numbers for the tip(1C) program. This file is normally unreadable and may contain privileged information. The format of the file is a series of lines of the form

system-name [\t] **phone-number*

The system name is one of those defined in the remote(4) file and the telephone number is constructed from any sequence of characters terminated only by a comma (,) or the end of the line. The = and * characters are indicators that inform the auto-call units to pause and wait for a second dial tone (when going through an exchange). The = is required by the DF02-AC and the * is required by the BIZCOMP 1030.

Only one telephone number per line is permitted. However, if more than one line in the file contains the same system name, tip(1C) will attempt to dial each one in turn, until it establishes a connection.

EXAMPLES

As distributed, the file /etc/phones contains a dummy entry. This should be replaced by a line (or lines) in the format described earlier. For example,

```
plato *5551234,
hegel *5551235,
```

FILES

/etc/phones

SEE ALSO

tip(1C), remote(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 display-able 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

`tplot`(1G), `plot`(3X), `term`(4).

WARNINGS

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

postscript — POSTSCRIPT print file format

DESCRIPTION

The POSTSCRIPT print file format is a programming language with powerful graphics primitives for describing printed pages. A growing number of devices which print POSTSCRIPT page descriptions are available. POSTSCRIPT printer include the Apple LaserWriter®, QMS PS-800, 1200, and 2400, Dataproducts LZR-2665 and 2660, and Linotype Linotronic 100 and 300 typesetters. The TRANSCRIPT package of UNIX software allows UNIX systems access to POSTSCRIPT printers.

The complete POSTSCRIPT language is described in the book

POSTSCRIPT Language Reference Manual
by Adobe Systems Incorporated
published by Addison-Wesley Publishing Company
ISBN 0-201-10174-2, 322 pages, illustrated
Library of Congress: QA76.73.P67P67 1985 005.13'3 85-15693

The Reference Manual provides a comprehensive presentation of of the language, its graphics, and its font facilities, including the precise semantics of every POSTSCRIPT operator. Also covered are a set of POSTSCRIPT file structuring conventions which are used by the TRANSCRIPT system components.

SEE ALSO

transcript(1M).

NAME

printcap — printer-capability database

SYNOPSIS

/etc/printcap

DESCRIPTION

printcap is a simplified version of the termcap(4) database used to describe line printers. The spooling system accesses the printcap file every time it is used, allowing dynamic addition and deletion of printers. Each entry in the database is used to describe one printer. This database may not be substituted, as is possible for termcap, because it may allow accounting to be bypassed.

The default printer is normally lp, though the environment variable PRINTER may be used to override this. Each spooling utility supports the flag option -P*printer* to allow explicit naming of a destination printer.

For a complete discussion on how setup the database for a given printer see *A/UX Local System Administration..*

CAPABILITIES

Refer to termcap(4) for a description of the file layout.

Name	Type	Default	Description
af	str	NULL	Name of accounting file.
br	num	none	If lp is a tty, set the baud rate (ioctl call).
cc	num	0	If lp is a tty, clear control flag bits (termio.h).
cf	str	NULL	cifplot data filter.
cs	num	0	Similar to cc, but set the bits.
df	str	NULL	Tex data filter (DVI format).
fd	bool	FALSE	If lp is a tty, use DTR/DCD flow control.
ff	str	"\f"	String to send for a form feed.
fo	bool	FALSE	Print a form feed when device is opened.
gf	str	NULL	Graph data filter (plot(3X) format).
hl	bool	FALSE	Print the burst header page last.
ic	num	0	If lp is a tty, clear input flag bits (termio.h) .
if	str	NULL	Name of text filter that does accounting.
is	num	0	Similar to ic, but set the bits.
lc	num	0	If lp is a tty, clear the local flag bits (termio.h).
lf	str	"/dev/console"	Error logging filename.
lo	str	"lock"	Name of lock file.
lp	str	"/dev/printer"	Device name to be opened for output.
ls	num	0	Similar to lc, but set the bits.

mx	num	1000	Maximum file size (in BUFSIZ blocks). Use 0 for unlimited size.
nd	str	NULL	Next directory for list of queues (unimplemented).
nf	str	NULL	Ditroff data filter (device independent troff).
oc	num	0	If `lp` is a tty, clear output flag bits (`termio.h`).
of	str	NULL	Name of the output filtering program.
os	num	0	Similar to `oc` but set the bits.
pc	num	200	Price per foot or page in hundredths of a cent.
pl	num	66	Page length (in lines).
pw	num	132	Page width (in characters).
px	num	0	Page width in pixels (horizontal).
py	num	0	Page length in pixels (vertical).
rf	str	NULL	Filter for printing FORTRAN-style text files.
rg	str	NULL	Restricted group. Only members of the group are allowed access.
rm	str	NULL	Machine name for remote printer.
rp	str	"lp"	Remote printer-name argument.
rs	bool	FALSE	Restrict remote users to those with local accounts.
rw	bool	FALSE	Open the printer device for reading and writing.
sb	bool	FALSE	Short banner (one line only).
sc	bool	FALSE	Suppress multiple copies.
sd	str	"/usr/spool/lpd"	Spool directory.
sf	bool	FALSE	Suppress form feeds.
sh	bool	FALSE	Suppress printing of burst page header.
st	str	"status"	Status filename.
tf	str	NULL	Troff data filter (cat phototypesetter).
tr	str	NULL	Trailer string to print when queue empties.
vf	str	NULL	Raster image filter.

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

FILTERS

The `lpd`(8) daemon creates a pipeline of *filters* to process files for various printer types. The filters selected depend on the flags passed to `lpr`(1). The pipeline set up is:

-p	pr | if	Regular text + `pr`(1)
none	if	Regular text
-c	cf	cifplot
-d	df	CVI (tex)
-g	gf	plot(3)
-n	nf	ditroff
-f	rf	Fortran
-t	tf	troff
-v	vf	Raster image

The `if` filter is invoked with arguments:

> `if [ -c ] -w`*width* `-l`*length* `-i`*indent* `-n` *login* `-h` *host acct-file*

The `-c` flag option is passed only if the `-l` flag option (pass control characters literally) is specified to `lpr`. The values of *width* and *length* specify the page width and length (from `pw` and `pl`, respectively) in characters. The `-n` and `-h` parameters specify the login name and the host name of the owner of the job, respectively. The value of *acct-file* is passed from the `af` `printcap` entry.

If no `if` filter is specified, the `of` filter is used instead, with the distinction that `of` is opened only once, while `if` is opened for every individual job. Thus, `if` is better suited to performing accounting. The `of` filter only has the *width* and *length* flag options.

All other filters are called as follows:

> `filter -x`*width* `-y`*length* `-n` *login* `-h` *host acct-file*

where *width* and *length* are represented in pixels, specified by the `px` and `py` entries, respectively.

All filters take `stdin` as the file and `stdout` as the printer, may log either to `stderr` or `syslog`(3), and must not ignore SIGINT.

ERRORS

Error messages generated by the line printer programs themselves (the `lp*` programs) are logged by `syslog`(3) using the *LPR* facility. Messages printed on `stderr` of one of the filters are sent to the corresponding `lf` file. The filters may, of course, use `syslog` themselves.

Error messages sent to the console have both a RETURN and a line feed appended to them, rather than just a line feed.

SEE ALSO

`termcap`(4), `lpc`(1m), `lpd`(1m), `pac`(1m), `lpr`(1), `lpq`(1), `lprm`(1).

NAME

`profile` — setting up an environment at login time

DESCRIPTION

If your login directory contains a file named `.profile`, that file will be executed (via the shell's `exec .profile`) before your session begins; `.profiles` are handy for setting exported environment variables and terminal modes. If the file `/etc/profile` exists, it will be executed for every user before the `.profile`. The following example is typical.

```
trap "" 1 2 3
TZ=`/bin/cat /etc/TIMEZONE`
PATH=/usr/lib/acct:/bin:/usr/bin
TERM=mac2
MAILCHECK=60
MAILPATH=/usr/mail/$LOGNAME
export LOGNAME TZ TERM PATH
readonly LOGNAME
umask 022
case "$0" in
-sh | -rsh)
        trap : 1 2 3
        cat /etc/motd
        trap "" 1 2 3
        if mail -e
        then
                echo "you have mail"
        fi
        if [ $LOGNAME != root ]
        then
                news -n
        fi
        ;;
-su)
        :
        ;;
esac
trap 1 2 3
stty susp '^Z'
stty erase DEL intr '^C'
stty ixon
```

FILES

`/etc/profile`
`$HOME/.profile`

SEE ALSO

env(1), login(1), mail(1), sh(1), stty(1), su(1), environ(5), term(5).

NAME

protocols — protocol name database

DESCRIPTION

The protocols file contains information regarding the known protocols used in the DARPA Internet. For each protocol a single line should be present with the following information:

official protocol name
protocol number
aliases

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

Protocol names may contain any printable character other than a field delimiter, newline, or comment character.

FILES

/etc/protocols

SEE ALSO

getprotoent(3N).

BUGS

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

NAME

`ptab` — partition table file

SYNOPSIS

`/etc/ptab`

DESCRIPTION

The `ptab` file contains information regarding the known partitions present on the local machine. It is read and/or modified by the `pname`(1M) utility. The system administrator can modify it with a text editor, though this is not recommended.

For each partition a single line should be present with the following information:

name Name of the partition — must not be greater than 32 characters long.

type Type of the partition — must not be greater than 32 characters long. If this field is left empty, the default type `Apple_UNIX_SVR2` will be assumed.

controller This is the controller number of the disk containing this partition.

disk This is the disk number (for the specified controller) of the disk containing this partition.

slice This is the slice (partition) number of the partition.

comment All additional information at the end of the line is treated as a comment.

The partition table file is an ASCII file. Fields within an entry are separated from eachother by colons. Each entry is separated from the next by a newline. Entries are separated by newlines. The `ptab` file can also have a line beginning with the sharp character (#), which means that this line should be treated as a comment and ignored.

EXAMPLES

Here is a sample `/etc/ptab` file:

```
#name:type:controller:disk:slice[:comment]
#root::0:0:0:assigned by default
#swap::0:0:1:assigned by default
src::0:0:3
users::1:0:0:on extra disk
Macintosh:Apple_HFS:0:0:13:Mac partition
```

FILES

/etc/ptab

SEE ALSO

dp(1M), pname(1M), getptabent(3).

WARNINGS

Appropriate precautions must be taken to lock the /etc/ptab file against simultaneous modifications.

BUGS

The current revision of the software will not support colons (:) in partition names or partition types.

NAME

`rcsfile` — format of an RCS file

DESCRIPTION

An RCS file is an ASCII file. Its content is described by the grammar below. The text is free format; that is, spaces, tabs, and newlines have no significance except in strings. Strings are enclosed by `@`. If a string contains an `@`, it must be doubled.

The metasyntax uses the following conventions: | (bar) separates alternatives; { and } enclose optional phrases; { and }* enclose phrases that may be repeated zero or more times; { and }+ enclose phrases that must appear at least once and may be repeated; nonterminal symbols are set in italic font, and literals are set in a constant-width font.

rcstext	::=	*admin* {*delta*}* *desc* {*deltatext*}*	
admin	::=	`head`	{*num*};
		`access`	{*id*}*;
		`symbols`	{*id* : *num*}*;
		`locks`	{*id* : *num*}*;
		`comment`	{*string*};
delta	::=	*num*	
		`date`	*num*;
		`author`	*id*;
		`state`	{*id*};
		`branches`	{*num*}*;
		`next`	{*num*};
desc	::=	`desc`	*string*
deltatext	::=	*num*	
		`log`	*string*
		`text`	*string*
num	::=	{*digit*{.}}+	
digit	::=	`0` | `1` | ... | `9`	
id	::=	*letter*{*idchar*}*	
letter	::=	`A` | `B` | ... | `Z` | `a` | `b` | ... | `z`	

idchar	::=	Any printing ASCII character except space, tab, carriage return, newline, and *special*.
special	::=	; | : | , | @
string	::=	@{any ASCII character, with @ doubled}*@

Identifiers are case sensitive. Keywords are in lowercase only. The sets of keywords and identifiers may overlap.

The *delta* nodes form a tree. All nodes whose numbers consist of a single pair (2.3, 2.1, 1.3, and so forth) are on the trunk and are linked through the next field in order of decreasing numbers. The head field in the *admin* node points to the head of that sequence which contains the highest pair.

All *delta* nodes whose numbers consist of $2n$ fields ($n \geq 2$) (3.1.1.1, 2.1.2.2, and so forth) are linked as follows. All nodes whose first $(2n)$-1 number fields are identical are linked through the next field in order of increasing numbers. For each such sequence, the *delta* node whose number is identical to the first $2(n$-1) number fields of the deltas on that sequence is called the branchpoint. The branches field of a node contains a list of the numbers of the first nodes of all sequences for which it is a branchpoint. This list is ordered in increasing numbers.

DISCLAIMER

This reference manual entry describes a utility that Apple understands to have been released into the public domain by its author or authors. Apple has included this public domain utility for your convenience. Use it at your own discretion. Often the source code can be obtained if additional requirements are met, such as the purchase of a site license from an author or institution.

IDENTIFICATION

Author: Walter F. Tichy, Purdue University, West Lafayette, IN 47907.

SEE ALSO

`ci`(1), `co`(1), `ident`(1), `rcs`(1), `rcsdiff`(1), `rcsintro`(1), `rcsmerge`(1), `rlog`(1), `sccstorcs`(1M).

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 */
  short    r_type ;    /* relocation type */
} ;


/*
 * All generics
 * reloc already performed to symbol in the
 * same section
 */
#define  R_ABS        0

/*
 * DEC Processors  VAX 11/780 and VAX 11/750
 *
 */
#define R_RELBYTE     017
#define R_RELWORD     020
#define R_RELLONG     021
#define R_PCRBYTE     022
#define R_PCRWORD     023
#define R_PCRLONG     024

/*
 * Motorola 68000 uses R_RELBYTE, R_RELWORD, R_RELLONG,
 * R_PCRBYTE, and R_PCRWORD as for DEC machines above.
 */
```

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 a symbol's virtual address.
R_RELWORD	A direct 16-bit reference to a symbol's virtual address.
R_RELLONG	A direct 32-bit reference to a symbol's virtual address.
R_PCRBYTE	A "PC-relative" 8-bit reference to a symbol's virtual address.
R_PCRWORD	A "PC-relative" 16-bit reference to a symbol's virtual address.
R_PCRLONG	A "PC-relative" 32-bit reference to a symbol's virtual address.

On the VAX processors, relocation of a symbol index of -1 indicates that the relative difference between the current segment's start address and the program's load address is added to the relocatable address.

Other relocation types will be defined as they are needed.

Relocation entries are generated automatically by the assembler and automatically utilized by the link editor. A link editor option exists for removing the relocation entries from an object file.

SEE ALSO

`ld`(1), `strip`(1), `a.out`(4), `syms`(4).

NAME

`remote` — remote host description file

SYNOPSIS

`/etc/remote`

DESCRIPTION

The systems known by `tip`(1C) and their attributes are stored in an ASCII file which is structured somewhat like the `termcap`(4) file. Each line in the file provides a description for a single *system*. Fields are separated by a colon (`:`). Lines ending in a \ character with a newline immediately following are continued on the next line.

The first entry is the name(s) of the host system. If there is more than one name for a system, the names are separated by vertical bars. Following the name of the system are the fields of the description. A field name followed by an = sign indicates that a string value follows. A field name followed by a # sign indicates a following numeric value.

Entries named `tip*` and `cu*` are used as default entries by `tip`, and the `cu` interface to `tip`, as follows: When `tip` is invoked with only a telephone number, it looks for an entry of the form `tip300`, where 300 is the baud rate with which the connection is to be made; when the `cu` interface is used, entries of the form `cu300` are used.

CAPABILITIES

Capabilities are either strings (`str`), numbers (`num`), or boolean flags (`bool`). A string capability is specified by *capability=value*; for example, `dv=/dev/harris`. A numeric capability is specified by *capability#value*; for example, `xa#99`. A boolean capability is specified by simply listing the capability.

`at` (*str*) Auto call unit type.

`br` (*num*) The baud used in establishing a connection to the remote host. This is a decimal number and the default is 300 baud.

`cm` (*str*) An initial connection message to be sent to the remote host. For example, if a host is reached through a port selector, this might be set to the appropriate sequence required to switch to the host.

`cu` (*str*) Call unit if making a telephone call. Default is the same as the `dv` field.

`di` (*str*) Disconnect message sent to the host when a disconnect is requested by the user.

`du` (*bool*) This host is on a dialup line.

`dv` (*str*) Device(s) to open to establish a connection. If this file refers to a terminal line, `tip`(1C) attempts to perform an exclusive open on the device to insure that only one user at a time has access to the port.

`el` (*str*) Characters marking an end-of-line. The default is NULL. The character ~ escapes are only recognized by `tip` after one of the characters in `el`, or after a return.

`fs` (*str*) Frame size for transfers. The default frame size is equal to `BUFSIZ`.

`hd` (*bool*) The host uses half-duplex communication; local echo should be performed.

`ie` (*str*) Input end-of-file marks. The default is NULL.

`mt` (*str*) Modem type (for use by `tip`). If `mt` is specified, the `at` field must appear as `at="generic"`. `tip` will then look in `/etc/dialup` for the appropriate modem escape sequences and call the generic dialup routine. If `mt` is not specified, `tip` will assume that it was compiled with the appropriate modem interface module

```
$(cc) -o tip -D${MODEM}
```

`oe` (*str*) Output end-of-file string. The default is NULL. When `tip` is transferring a file, this string is sent at end-of-file.

`pa` (*str*) The type of parity to use when sending data to the host. This may be one of `even`, `odd`, `none`, `zero` (always set bit 8 to zero), or `one` (always set bit 8 to 1). The default is even parity.

`pn` (*str*) Telephone number(s) for this host. If the telephone number field contains an `@` sign, `tip` searches the `/etc/phones` file for a list of telephone numbers (see `phones`(4)).

`tc` (*str*) Indicates that the list of capabilities is continued in the named description. This is used primarily to share common capability information.

Here is a short example showing the use of the capability continuation feature

```
UNIX-1200:\
:dv=/dev/cua0:el=^D^U^C^S^Q^O@:du:at=ventel:ie=#$%:\
:oe=^D:br#1200:
arpavax|ax:\
:pn=7654321%:tc=UNIX-1200
```

FILES

/etc/remote

SEE ALSO

tip(1C), phones(4).

NAME

`resolver` — `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 is designed to be human readable and contains a list of name-value pairs that provide various types of resolver information.

On a normally configured system this file should not be necessary. The only name server to be queried will be on the local machine and the domain name is retrieved from the system.

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, in that case the resolver library queries tries them 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).

domain
: followed by a 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`(2N) is used (everything after the first "."). Finally, if the host name does not contain a domain part, the root domain is assumed.

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

FILES

`/etc/resolv.conf`

SEE ALSO

named(1M), gethostbyname(3N), resolver(3N).

NAME

rhosts — trusted hosts file format

DESCRIPTION

The login directory for each user can contain a .rhosts file that enumerates remote hosts having equivalent account names. (The hosts names must be the standard names as described in remsh(1N)).

Each line in this file should contain a *rhost* and a *username* separated by a space, allowing additional cases where logins without passwords are to be permitted.

When you rlogin as the same user on an equivalent host, you don't need to give a password.

To avoid security problems, the .rhosts file must be owned by either the remote user or root. Note that, for security reasons, root is an exception to the above; a superuser on an equivalent host must still supply the password to remotely login as root unless the root account has its own private equivalence list in a file .rhosts in the root directory. Note that a .rhosts file for the root account is not recommended where secure systems are required.

Your remote terminal type is the same as your local terminal type (as given in your environment TERM variable). See rlogin(1N) for other details concerning the line discipline and escape characters.

FILES

/home-directory/.rhosts

SEE ALSO

remsh(1N), rlogin(1N).

NAME

rmtab — remotely mounted file system table

DESCRIPTION

rmtab resides in directory /etc and contains a record of all clients that have done remote mounts of file systems from this machine. Whenever a remote mount is done, an entry is made in the rmtab file of the machine serving up that file system. umount removes entries. umount -a broadcasts to all servers, and informs them that they should remove all entries from rmtab created by the sender of the broadcast message. By placing a umount -a command in /etc/sysinitrc, rmtab tables can be purged of entries made by a crashed host, which upon rebooting did not remount the same file systems it had before. 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(1M) and shutdown(1M).

FILES

/etc/rmtab

SEE ALSO

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

BUGS

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

NAME

rpc — RPC program number database

SYNOPSIS

/etc/rpc

DESCRIPTION

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

server-name program-number [*alias...*]

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

EXAMPLES

```
#
#       rpc   1.1   86/07/07
#
portmapper      100000  portmap sunrpc
rstatd          100001  rstat rup perfmeter
rusersd         100002  rusers
nfs             100003  nfsprog
ypserv          100004  ypprog
mountd          100005  mount showmount
ypbind          100007
walld           100008  rwall shutdown
yppasswdd       100009  yppasswd
etherstatd      100010  etherstat
rquotad         100011  rquotaprog quota rquota
sprayd          100012  spray
3270_mapper     100013
rje_mapper      100014
selection_svc   100015  selnsvc
database_svc    100016
rexd            100017  rex
alis            100018
sched           100019
llockmgr        100020
nlockmgr        100021
x25.inr         100022
statmon         100023
status          100024
```

FILES

/etc/rpc

SEE ALSO

rpc(3N).

NAME

`sccsfile` — format of an SCCS file

DESCRIPTION

An SCCS 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 ID's 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 numeric group ID's of users who may add deltas to the file, separated by newlines. The lines containing these login names and/or numeric group ID's 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 (e.g., 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:

```
@I DDDDD
@D DDDDD
@E DDDDD
```

respectively. The digit string is the serial number corresponding to the delta for the control line.

SEE ALSO

`admin`(1), `cdc`(1), `comb`(1), `delta`(1), `get`(1), `help`(1), `rmdel`(1), `sact`(1), `sccs`(1), `sccsdiff`(1), `unget`(1), `val`(1), `what`(1),
"SCCS Reference" in *A/UX Programming Languages and Tools, Volume 2.*

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[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 */
} ;
```

File pointers are byte offsets into the file; they can be used as the offset in a call to fseek(3S). 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

servers — Internet server database

DESCRIPTION

The servers file contains the list of servers that inetd(1M) operates. For each server a single line should be present with the following information:

name of server
protocol
server location

If the server is RPC-based, then the name field should be rpc, and following the server location are two additional fields, one with the RPC program number, the second with either a version number or a range of version numbers.

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

The name of the server should be the official service name as contained in services(4N). The protocol entry is either udp or tcp. The server location is the full pathname of the server program.

EXAMPLES

```
tcp     tcp   /usr/etc/in.tcpd
telnet  tcp   /usr/etc/in.telnetd
shell   tcp   /etc/in.rshd
login   tcp   /etc/in.rlogind
exec    tcp   /usr/etc/in.rexecd
tcp     udp   /usr/etc/in.ttcpd
syslog  udp   /usr/etc/in.syslog
comsat  udp   /usr/etc/in.comsat
talk    udp   /usr/etc/in.talkd
time    tcp   /usr/etc/in.timed
rpc     udp   /usr/etc/rpc.rstatd    100001   1-2
rpc     udp   /usr/etc/rpc.rusersd   100002   1
rpc     udp   /usr/etc/rpc.rwalld    100008   1
rpc     udp   /usr/etc/rpc.mountd    100005   1
```

FILES

/etc/servers

SEE ALSO

inetd(1M), services(4N).

BUGS

Because of a limitation on the number of open files, this file must contain fewer than 27 lines.

NAME

services — service name database

DESCRIPTION

The services file contains information regarding the known services available in the DARPA Internet. For each service a single line should be present with the following information:

official service name
port number
protocol name
aliases

Items are separated by any number of blanks or tab characters. The port number and protocol name are considered a single item; a / is used to separate the port and protocol (for example, 512/tcp). A # indicates the beginning of a comment; characters up to the end of the line are not interpreted by routines which search the file.

Service names may contain any printable character other than a field delimiter, newline, or comment character.

FILES

/etc/services

SEE ALSO

getservent(3N).

BUGS

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

NAME

slip.config — list of slip interfaces supported by a slip server

SYNOPSIS

/etc/slip.config

DESCRIPTION

The slip.config file must be configured on the slip server to establish slip connections between the slip client and slip host. slip(1M) is a program that assigns a tty line to a network interface for a point-to-point TCP/IP link.

Only the system administrator of the slip server can modify the /etc/slip.config file, which contains the slip server host address for each of the slip interfaces supported by the slip server. mkslipuser(1M) must then be executed to create the machine-readable slip.user file from the slip.config data file. A sample slip.config configuration file is

```
# slip.config configuration file
# Each line configures a serial line
#
128.120.254.3
128.120.254.3
```

In this example, the host has two serial interfaces available for slip use.

SEE ALSO

netstat(1), dslipuser(1M), ifconfig(1M), mkslipuser(1M), slip(1M) slip.hosts(4), slip.user(4).

NAME

slip.hosts — map user names to host addresses of slip client

SYNOPSIS

/etc/slip.hosts

DESCRIPTION

The slip.hosts file must be configured on the slip server to establish slip connections between the slip client and the slip host. slip(1M) is a program that assigns a tty line to a network interface for a point-to-point TCP/IP link.

Only the system administrator of the slip host can modify the /etc/slip.hosts file, which contains the Internet address and user name for each user with a slip connection to the slip server. A sample slip.hosts file is

```
# dialup slip.hosts table
# maps usercodes to host addresses
#
128.120.253.1 joe
128.120.253.2 chris
128.120.253.3 mike
128.120.253.4 linda
```

The Internet address in the first field is to be used when the user specified in the second field invokes slip.

SEE ALSO

netstat(1), dslipuser(1M), ifconfig(1M), mkslipuser(1M), slip(1M), slip.config(4), slip.user(4).

NAME

slip.user — user file created by mkslipuser

SYNOPSIS

/etc/slip.user

DESCRIPTION

The slip.user file must be configured on the slip server to establish slip connections between the slip client and slip host. slip(1M) is a program that assigns a tty line to a network interface for a point-to-point TCP/IP link.

The slip user file /etc/slip.user is not human readable and is generated by the command mkslipuser(1M). You can use the command dslipuser(1M) to display the contents of the user file, which reports the number of slip users on the system and the number of available slip interfaces.

SEE ALSO

netstat(1), dslipuser(1M), ifconfig(1M), mkslipuser(1M), slip(1M), slip.config(4), slip.hosts(4).

NAME

svfs — format of a System V system volume

SYNOPSIS

```
#include <sys/types.h>
#include <sys/param.h>
#include <svfs/filsys.h>
```

DESCRIPTION

Every SVFS file-system storage volume has a common format for certain vital information. Each volume is divided into a certain number of 512-byte sectors. Sector 0 contains the disk partition map. See dpme(4) for further information on its structure.

Sector 1 is the superblock. The format of a superblock is

```
/*
 * Structure of the superblock
 */

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 inodes
                                 in s_inode */
    ino_t   s_inode[NICINOD]; /* free inode list */
    char    s_flock;          /* lock during free
                                 list manipulation */
    char    s_ilock;          /* lock during i-list
                                 manipulation */
    char    s_fmod;           /* superblock modified
                                 flag */
    char    s_ronly;          /* mounted read-only
                                 flag */
    time_t  s_time;           /* last superblock
                                 update */
    short   s_dinfo[4];       /* device information */
    daddr_t s_tfree;          /* total free blocks */
    ino_t   s_tinode;         /* total free inodes */
    char    s_fname[6];       /* file-system name */
    char    s_fpack[6];       /* file-system pack name */
    long    s_fill[13];       /* ADJUST size of
                                 filsys to 512 */
    long    s_state;              /* file-system state */
    ino_t   s_lasti;          /* start place for
```

```
                                        circular search */
    ino_t    s_nbehind;              /* est # free inodes
                                        before s_lasti */
    long     s_magic;                /* magic number to
                                        indicate new filesys */
    long     s_type;                 /* type of new filesys */
};

#define   FsMAGIC  0xfd187e20    /* s_magic number */
#define   Fs1b     1             /* 512-byte block */
#define   Fs2b     2             /* 1024-byte block */
#define   Fs4b     4             /* 2048-byte block */
```

`s_type` indicates the file-system type. Currently, two types of file systems are supported: the original 512-byte block system and the new improved 1024-byte block system. `s_magic` is used to distinguish the original 512-byte block file systems from the newer file systems. If this field is not equal to the magic number, `FsMAGIC`, the type is assumed to be `Fs1b`; otherwise, the `s_type` field is used. In the following description, a block is then determined by the type. For the original 512-byte block file system, a block is 512 bytes. For the 1024-byte block file system, a block is 1024 bytes or two sectors. The operating system takes care of all conversions from logical block numbers to physical sector numbers.

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

The free list for each volume is maintained as follows. The `s_free` array contains, in `s_free[1]...` `s_free[s_nfree-1]`, up to 49 numbers of free blocks. `s_free[0]` is the block number of the head of a chain of blocks constituting the free list. The first long in each free-chain block is the number (up to 50) of free-block numbers listed in the next 50 longs of this chain member. The first of these 50 blocks is the link to the next member of the chain.

To allocate a block, decrement s_nfree, and the new block is s_free[s_nfree]. If the new block number is 0, there are no blocks left, so give an error. If s_nfree became 0, read in the block named by the new block number, replace s_nfree by its first word, and copy the block numbers in the next 50 longs into the s_free array.

To free a block, check if s_nfree is 50. If so, copy s_nfree and the s_free array into it, write it out, and set s_nfree to 0. In any event, set s_free[s_nfree] to the number of the freed block and increment s_nfree.

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

s_ninode is the number of free inumbers in the s_inode array. To allocate an inode, if s_ninode is greater than 0, decrement it and return s_inode[s_ninode]. To allocate an inode, if s_ninode is 0, read the i-list, place the numbers of all free inodes (up to 100) into the s_inode array, and then try again. To free an inode, 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 inode into any table. This list of inodes is only to speed up the allocation process. The information as to whether the inode is really free or not is maintained in the inode itself.

s_tinode is the total free inodes available in the file system.

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

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

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

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

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

FILES

`/usr/include/svfs/filsys.h`
`/usr/include/sys/stat.h`

SEE ALSO

`fsck`(1M), `fsdb`(1M), `mkfs`(1M), dpme(4), `ufs`(4), `inode`(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.

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

Filename 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

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

struct    syment
{
 union                          /* ways to get a
                                   symbol name */
 {
   char   _n_name[SYMNMLEN] ;  /* names less than
                                   8 chars */
```

```
   struct                          /* names 8 char
                                      or more */
   {
      long         _n_zeroes;    /* == 0L when in
                                    string table */
      long         _n_offset;    /* location of name in
                                    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 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 is as 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
        {
                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).
"COFF Reference" in *A/UX Programming Languages and Tools, Volume 2.*

WARNINGS

In machines in which a long are equivalent to an int (M68000 and VAX), the long is converted to int in the compiler to minimize the complexity of the compiler code generator. Thus, the information about which symbols are declared as long and which as int cannot be determined from the symbol table.

NAME

tar — format of tar header

DESCRIPTION

tar saves and restores files on magnetic tape or floppy disks. The tar header format is as follows:

```
# define TBLOCK 512
# define NBLOCK 40
# define NAMSIZ 100
union hblock {
         char dummy[TBLOCK];
         struct header {
                  char name[NAMESIZ];
                  char mode[8];
                  char uid[8];
                  char gid[8];
                  char size[12];
                  char mtime[12];
                  char chksum[8];
                  char linkflag;
                  char linkname[NAMESIZ];
        } dbuf;
} dblock, tbuf[NBLOCK];
```

SEE ALSO

tar(1).

NAME

term — format of compiled term file

SYNOPSIS

term

DESCRIPTION

Compiled terminfo descriptions are placed under the directory /usr/lib/terminfo. In order to avoid a linear search of a huge A/UX 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 or more bit byte is assumed, but no assumptions about byte ordering or sign extension are made.

The compiled file is created with the tic(1M) program, and read by the routine setupterm. Both of these pieces of software are part of 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.

Short integers are stored in two 8-bit bytes. The first byte contains the least significant 8 bits of the value, and the second byte contains the most significant 8 bits. (Thus, the value represented is 256*second+first.) The value −1 is represented by 0377, 0377, other negative value are illegal. The −1 generally means that a capability is missing from this terminal. Computers where this does not correspond to the hardware read the integers as two bytes and compute the result.

The terminal names section comes next. It contains the first line of the terminfo description, listing the various names for the terminal, separated by the ''|'' character. The section is terminated with an ASCII NUL character.

The boolean flags have one byte for each flag. This byte is either 0 or 1 as the flag is present or absent. The capabilities are in the same order as the file `<term.h>`.

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

The numbers section is similar to the flags section. Each capability takes up two bytes, and is stored as a short integer. If the value represented is −1, the capability is taken to be missing.

The strings section is also similar. Each capability is stored as a short integer, in the format above. A value of −1 means the capability is missing. Otherwise, the value is taken as an offset from the beginning of the string table. Special characters in CONTROL-*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. The name field cannot exceed 128 bytes.

FILES

`/usr/lib/terminfo/*/*`
compiled terminal capability data base

SEE ALSO

`curses`(3X), `terminfo`(4).

NAME

termcap — terminal capability database

SYNOPSIS

/etc/termcap

DESCRIPTION

termcap is a data base describing terminals used, for example by vi(1). Terminals are described in termcap by giving a set of capabilities which they have and by describing how operations are performed. Padding requirements and initialization sequences are included in termcap.

Entries in termcap consist of a number of colon (:) separated fields. The first entry for each terminal gives the names which are known for the terminal, separated by | characters. The first name is always 2 characters long and is used by older version 6 systems which store the terminal type in a 16 bit word in a systemwide data base. The second name given is the most common abbreviation for the terminal, and the last name given should be a long name fully identifying the terminal. The second name should contain no blanks; the last name may well contain blanks for readability.

CAPABILITIES

(P) indicates padding may be specified

(P*) indicates that padding may be based on no. lines affected

Name	Type	Pad?	Description
ae	str	(P)	End alternate character set
al	str	(P*)	Add new blank line
am	bool		Terminal has automatic margins
as	str	(P)	Start alternate character set
bc	str		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*)	Clear to end of display
ce	str	(P)	Clear to end of line
ch	str	(P)	Like cm but horizontal motion only, line stays same
cl	str	(P*)	Clear screen
cm	str	(P)	Cursor motion
co	num		Number of columns in a line
cr	str	(P*)	Carriage return, (default CONTROL-M)

cs	str	(P)	Change scrolling region (vt100), like cm
cv	str	(P)	Like ch but vertical only.
da	bool		Display may be retained above
dB	num		Number of millisec of bs delay needed
db	bool		Display may be retained below
dC	num		Number of millisec of cr delay needed
dc	str	(P*)	Delete character
dF	num		Number of millisec of ff delay needed
dl	str	(P*)	Delete line
dm	str		Delete mode (enter)
dN	num		Number of millisec of nl delay needed
do	str		Down one line
dT	num		Number of millisec of tab delay needed
ed	str		End delete mode
ei	str		End insert mode; give :ei=: if ic
eo	str		Can erase overstrikes with a blank
ff	str	(P*)	Hardcopy terminal page eject (default CONTROL-L)
hc	bool		Hardcopy terminal
hd	str		Half-line down (forward 1/2 linefeed)
ho	str		Home cursor (if no cm)
hu	str		Half-line up (reverse 1/2 linefeed)
hz	str		Hazeltine; can't print ~'s
ic	str	(P)	Insert character
if	str		Name of file containing is
im	str		Insert mode (enter); give :im=: if ic
in	bool		Insert mode distinguishes nulls on display
ip	str	(P*)	Insert pad after character inserted
is	str		Terminal initialization string
k0-k9	str		Sent by "other" function keys 0-9
kb	str		Sent by backspace key
kd	str		Sent by terminal down arrow key
ke	str		Out of "keypad transmit" mode
kh	str		Sent by home key
kl	str		Sent by terminal left arrow key
kn	num		Number of "other" keys
ko	str		Termcap entries for other non-function keys
kr	str		Sent by terminal right arrow key
ks	str		Put terminal in "keypad transmit" mode
ku	str		Sent by terminal up arrow key
l0-l9	str		Labels on "other" function keys
li	num		Number of lines on screen or page
ll	str		Last line, first column (if no cm)

ma	str		Arrow key map, used by vi version 2 only
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*)	Newline character (default \n)
ns	bool		Terminal is a CRT but doesn't scroll.
os	bool		Terminal overstrikes
pc	str		Pad character (rather than null)
pt	bool		Has hardware tabs (may need to be set with is)
se	str		End stand out mode
sf	str	(P)	Scroll forwards
sg	num		Number of blank chars left by so or se
so	str		Begin stand out mode
sr	str	(P)	Scroll reverse (backwards)
ta	str	(P)	Tab (other than CONTROL-I or with padding)
tc	str		Entry of similar terminal - must be last
te	str		String to end programs that use cm
ti	str		String to begin programs that use cm
uc	str		Underscore one char and move past it
ue	str		End underscore mode
ug	num		Number of blank chars left by us or ue
ul	bool		Terminal underlines even though it doesn't overstrike
up	str		Upline (cursor up)
us	str		Start underscore mode
vb	str		Visible bell (may not move cursor)
ve	str		Sequence to end open/visual mode
vs	str		Sequence to start open/visual mode
xb	bool		Beehive (f1=escape, f2=ctrl C)
xn	bool		A newline is ignored after a wrap (Concept)
xr	bool		Return acts like ce \r \n (Delta Data)
xs	bool		Standout not erased by writing over it (HP 264?)
xt	bool		Tabs are destructive, magic so char (Teleray 1061)

A Sample Entry

The following entry, which describes the Concept–100, is among the more complex entries in the termcap file as of this writing. (This particular concept entry is outdated and is used as an example only.)

```
cl c100 concept100:is=\EU\Ef\E7\E5\E8\El\ENH\EK\E\200\Eo&\200:\
:al=3*\E^R:am:bs:cd=16*\E^C:ce=16\E^S:cl=2*^L\
:cm=\Ea%+ %+ :co#80:\ :dc=16\E^A:dl=3*\E^B\
:ei=\E\200:eo:im=\E^P:in:ip=16*:li#24:mi:nd=\E=:\
:se=\Ed\Ee:so=\ED\EE:ta=8\t:ul:up=\E;:vb=\Ek\EK:xn:
```

Entries may continue onto multiple lines by giving a \ as the last character of a line, and that empty fields may be included for readability (here between the last field on a line and the first field on the next). Capabilities in `termcap` are of three types: Boolean capabilities which indicate that the terminal has some particular feature, numeric capabilities giving the size of the terminal or the size of particular delays, and string capabilities, which give a sequence which can be used to perform particular terminal operations.

Types of Capabilities

All capabilities have two letter codes. 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 `co` which indicates the number of columns the terminal has gives the value "80" for the Concept.

Finally, string valued capabilities, such as `ce` (clear to end of line sequence) are given by the two character code, an "=", and then a string ending at the next following ":". A delay in milliseconds may appear after the "=" in such a capability, and padding characters are supplied by the editor after the remainder of the string is sent to provide this delay. The delay can be either a integer, e.g., "20", or an integer followed by an "*", i.e. "3*". 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. When a "*" is specified, it is sometimes useful to give a delay of the form "3.5" specify a delay per unit to tenths of milliseconds.

A number of escape sequences are provided in the string valued capabilities for easy encoding of characters there. A `\E` maps to an escape character, CONTROL-X maps to a CONTROL-*x* for any appropriate *x*, and the sequences `\n`, `\r`, `\tr`, `\b`, and `\f` give a newline, return, tab, backspace and form feed. Finally, characters may be given as three octal digits after a `\`, and the characters ^ and `\` may be given as `\^` and `\\`. If it is necessary to place a `:` in a capability it must be escaped in octal as `\072`. If it is necessary to place a null character in a string capability it must be encoded as `\200`. The routines which deal with `termcap` use C strings, and strip the high bits of the output very late so that a `\200` comes out as a `\000` would.

Preparing Descriptions

We now outline how to prepare descriptions of terminals. The most effective way to prepare a terminal description is by imitating the description of a similar terminal in `termcap` and to build up a description gradually, using partial descriptions with `ex` to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the `termcap` file to describe it or bugs in `ex`. To easily test a new terminal description you can set the environment variable `TERMCAP` to a pathname of a file containing the description you are working on and the editor will look there rather than in `/etc/termcap`. `TERMCAP` can also be set to the `termcap` entry itself to avoid reading the file when starting up the editor. (This only works on version 7 systems.)

Basic Capabilities

The number of columns on each line for the terminal is given by the `co` numeric capability. If the terminal is a CRT, then the number of lines on the screen is given by the `li` 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, then this is given by the `cl` string capability. If the terminal can backspace, then it should have the `bs` capability, unless a backspace is accomplished by a character other than CONTROL-H, in which case you should give this character as the `bc` string capability. If it overstrikes (rather than clearing a position when a character is struck over) then it should have the `os` capability.

A very important point here is that the local cursor motions encoded in `termcap` are undefined at the left and top edges of a CRT terminal. The editor will never attempt to backspace around the left edge, nor will it attempt to go up locally off the top. The editor assumes that feeding off the bottom of the screen will cause the screen to scroll up, and the `am` capability tells whether the cursor sticks at the right edge of the screen. If the terminal has switch selectable automatic margins, the `termcap` file usually assumes that this is on, i.e. `am`.

These capabilities suffice to describe hardcopy and "glass-tty" terminals. Thus the model 33 teletype is described as

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

while the Lear Siegler ADM-3 is described as

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

Cursor Addressing

Cursor addressing in the terminal is described by a `cm` string capability, with `printf`(3S) like escapes `%x` in it. These substitute to encodings of the current line or column position, while other characters are passed through unchanged. If the `cm` string is thought of as being a function, then its arguments are the line and then the column to which motion is desired, and the `%` encodings have the following meanings:

`%d`	as in `printf`, 0 origin
`%2`	like %2d
`%3`	like %3d
`%.`	like %c
`%+`*x*	adds *x* to value, then %.
`%>`*xy*	if value > *x* adds *y*, no output.
`%r`	reverses order of line and column, no output
`%i`	increments line/column (for 1 origin)
`%%`	gives a single %
`%n`	exclusive or row and column with 0140 (DM2500)
`%B`	BCD `(16*(x/10)) + (x%10)`, no output.
`%D`	Reverse coding `(x-2*(x%16))`, no output. (Delta Data).

Consider the HP2645, 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 printed as two digits. Thus its `cm` capability is "`cm=6\E&%r%2c%2Y`." The Microterm ACT-IV needs the current row and column sent preceded by a CONTROL-T, with the row and column simply encoded in binary, "`cm=CONTROL-T%.%.`". Terminals which use "`%.`" need to be able to backspace the cursor (`bs` or `bc`), and to move the cursor up one line on the screen (`up` introduced below). This is necessary because it is not always safe to transmit `\t`, `\n`, CONTROL-D, and `\r`, as the system may change or discard them.

A final example is the LSI ADM-3a, which uses row and column offset by a blank character, thus

```
cm=\E=%+ %+
```

Cursor Motions

If the terminal can move the cursor one position to the right, leaving the character at the current position unchanged, then this sequence should be given as `nd` (nondestructive space). If it can move the cursor up a line on the screen in the same column, this should be given as `up`. If the terminal has no cursor addressing capability, but can home the cursor (to very upper left corner of screen) then this can be given as `ho`; similarly a fast way of getting to the lower left hand corner can be given as `ll`; this may involve going up with `up` from the home position, but the editor will never do this itself (unless `ll` does) because it makes no assumption about the effect of moving up from the home position.

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 `ce`. If the terminal can clear from the current position to the end of the display, then this should be given as `cd`. The editor only uses `cd` from the first column of a line.

Insert/Delete Line

If the terminal can open a new blank line before the line where the cursor is, this should be given as `al`; 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 `dl`; this is done only from the first position on the line to be deleted. If the terminal can scroll the screen backwards, then this can be given as `sb`, but just `al` suffices. 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 let the editor understand that deleting a line on the screen may bring non-blank lines up from below or that scrolling back with `sb` may bring down non-blank lines.

Insert/Delete Character

There are two basic kinds of intelligent terminals with respect to insert/delete character which can be described using `termcap`. 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 find out which 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''. If your terminal does something different and unusual then you may have to modify the editor to get it to use the insert mode your terminal defines. We have seen no terminals which have an insert mode not falling into one of these two classes.

The editor can handle 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 im the sequence to get into insert mode, or give it an empty value if your terminal uses a sequence to insert a blank position. Give as ei the sequence to leave insert mode (give this, with an empty value also if you gave im so). Now give as ic any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give ic, terminals which send a sequence to open a screen position should give it here. (Insert mode is preferable to the sequence to open a position on the screen if your terminal has both.) If post insert padding is needed, give this as a number of milliseconds 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.

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

Finally, you can specify delete mode by giving `dm` and `ed` to enter and exit delete mode, and `dc` to delete a single character while in delete mode.

Highlighting, Underlining, and Visible Bells

If your terminal has sequences to enter and exit standout mode, these can be given as `so` and `se`, respectively. If there are several flavors of standout mode (such as inverse video, blinking, or underlining; half bright is not usually an acceptable "standout" mode unless the terminal is in inverse video mode constantly) the preferred mode is inverse video by itself. 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 `ug` should be given to tell how many spaces are left.

Codes to begin underlining and end underlining can be given as `us` and `ue` respectively. If the terminal has a code to underline the current character and move the cursor one space to the right, such as the Microterm Mime, this can be given as `uc`. (If the underline code does not move the cursor to the right, give the code followed by a nondestructive space.)

Many terminals, such as the HP 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.

If the terminal has a way of flashing the screen to indicate an error quietly (a bell replacement) then this can be given as `vb`; it must not move the cursor. If the terminal should be placed in a different mode during open and visual modes of *ex*, this can be given as `vs` and `ve`, sent at the start and end of these modes respectively. These can be used to change, e.g., 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, the codes to enter and exit this mode can be given as `ti` and `te`. 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.

If your terminal correctly generates underlined characters (with no special codes needed) even though it does not overstrike, then you should give the capability `ul`. If overstrikes are erasable with a blank, then this should be indicated by giving `eo`.

Keypad

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 HP 2621 keys). If the keypad can be set to transmit or not transmit, give these codes as `ks` and `ke`. 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 `kl`, `kr`, `ku`, `kd`, and `kh` respectively. If there are function keys such as f0, f1, ..., f9, the codes they send can be given as `k0`, `k1`, ..., `k9`. If these keys have labels other than the default f0 through f9, the labels can be given as `l0`, `l1`, ..., `l9`. If there are other keys that transmit the same code as the terminal expects for the corresponding function, such as clear screen, the `termcap` 2 letter codes can be given in the `ko` capability, for example, "`:ko=cl,ll,sf,sb:`", which 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` entries.

The `ma` entry is also used to indicate arrow keys on terminals which have single character arrow keys. It is obsolete but still in use in version 2 of `vi`, which must be run on some minicomputers due to memory limitations. This field is redundant with `kl`, `kr`, `ku`, `kd`, and `kh`. It consists of groups of two characters. In each group, the first character is what an arrow key sends, the second character is the corresponding `vi` command. These commands are `h` for `kl`, `j` for `kd`, `k` for `ku`, `l` for `kr`, and `H` for `kh`. For example, the mime would be `:ma=^Kj^Zk^Xl:` indicating arrow keys left (CONTROL-h), down (CONTROL-K), up (CONTROL-Z), and right (CONTROL-X). (There is no home key on the mime.)

Miscellaneous

If the terminal requires other than a null (zero) character as a pad, then this can be given as `pc`.

If tabs on the terminal require padding, or if the terminal uses a character other than CONTROL-I to tab, then this can be given as `ta`.

Hazeltine terminals, which don't allow "~" characters to be printed should indicate `hz`. Datamedia terminals, which echo carriage-return linefeed for carriage return and then ignore a following linefeed should indicate `nc`. Early Concept terminals, which ignore a linefeed immediately after an `am` wrap, should indicate `xn`. If an erase-eol is required to get rid of standout (instead of merely writing on top of it), `xs` should be given. Teleray terminals, where tabs turn all characters moved over to blanks, should indicate `xt`. Other specific terminal problems may be corrected by adding more capabilities of the form *xx*.

Other capabilities include `is`, an initialization string for the terminal, and `if`, the name of a file containing long initialization strings. These strings are expected to properly clear and then set the tabs on the terminal, if the terminal has settable tabs. If both are given, `is` will be printed before `if`. This is useful where `if` is `/usr/lib/tabset/std` but `is` clears the tabs first.

Similar Terminals

If there are two very similar terminals, one can be defined as being just like the other with certain exceptions. The string capability `tc` can be given with the name of the similar terminal. This capability must be *last* and the combined length of the two entries must not exceed 1024. Since `termlib` routines search the entry from left to right, and since the tc capability is replaced by the corresponding entry, the capabilities given at the left override the ones in the similar terminal. A capability can be cancelled with *xx@* where *xx* is the capability. For example, the entry:

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

defines a "2621nl" that does not have the `ks` or `ke` capabilities, and hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

FILES

`/etc/termcap` file containing terminal descriptions

SEE ALSO

`ex`(1), `tset`(1), `ul`(1), `vi`(1), `termcap`(3X).

BUGS

`ex` allows only 256 characters for string capabilities, and the routines in `termcap(3X)` do not check for overflow of this buffer. The total length of a single entry (excluding only escaped newlines) may not exceed 1024.

The `ma`, `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.

NAME

terminfo — terminal capability database

SYNOPSIS

/usr/lib/terminfo/*/*

DESCRIPTION

terminfo is a data base describing terminals, used for example by vi(1) and curses(3X). Terminals are described in terminfo by giving a set of capabilities which they have, and by describing how operations are performed. Padding requirements and initialization sequences are included in terminfo.

Entries in terminfo consist of a number of "," separated fields. White space after each "," is ignored. The first entry for each terminal gives the names which are known for the terminal, separated by "|" characters. The first name given is the most common abbreviation for the terminal, 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 be in lower case and contain no blanks; the last name may well contain upper case and 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, thus "hp2621". This name should not contain hyphens, except that synonyms may be chosen that do not conflict with other names. Modes that the hardware can be in, or user preferences, should be indicated by appending a hyphen and an indicator of the mode. Thus, a vt100 in 132 column mode would be vt100-w.

The following suffixes should be used where possible:

Suffix	Meaning	Example
-w	Wide mode (more than 80 columns)	vt100-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
-*n*p	Number of pages of memory	c100-4p
-rv	Reverse video	c100-rv

CAPABILITIES

The variable is the name by which the programmer (at the `terminfo` level) accesses the capability. The capname is the short name used in the text of the database, and is used by a person updating the database. The i.code is the two letter internal code used in the compiled database, and always 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 and to allow the tabs in the source file `caps` to line up nicely. 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.

(P) indicates that padding may be specified

(G) indicates that the string is passed through `tparm` with parms as given (#*i*).

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

($\#_i$) indicates the *i*th parameter.

Variable Booleans	Capname	I. Code	Description
auto_left_margin,	bw	bw	cub1 wraps from column 0 to last column
auto_right_margin,	am	am	Terminal has automatic margins
beehive_glitch,	xsb	xb	Beehive (f1=escape, f2=ctrl C)
ceol_standout_glitch,	xhp	xs	Standout not erased by overwriting (hp)
eat_newline_glitch,	xenl	xn	newline ignored after 80 cols (Concept)
erase_overstrike,	eo	eo	Can erase overstrikes with a blank
generic_type,	gn	gn	Generic line type (e.g.,, dialup, switch).
hard_copy,	hc	hc	Hardcopy terminal
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
over_strike,	os	os	Terminal overstrikes
status_line_esc_ok,	eslok	es	Escape can be used on the status line
teleray_glitch,	xt	xt	Tabs ruin, magic so char (Teleray 1061)
tilde_glitch,	hz	hz	Hazeltine; can not print ~'s
transparent_underline,	ul	ul	underline character overstrikes
xon_xoff,	xon	xo	Terminal uses xon/xoff handshaking

Numbers:

columns,	cols	co	Number of columns in a line
init_tabs,	it	it	Tabs initially every # spaces
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 of blank chars left by smso or rmso
padding_baud_rate,	pb	pb	Lowest baud where cr/nl padding is needed
virtual_terminal,	vt	vt	Virtual terminal number (UNIX system)
width_status_line,	wsl	ws	No. columns in status line

Strings:

back_tab,	cbt	bt	Back tab (P)
bell,	bel	bl	Audible signal (bell) (P)
carriage_return,	cr	cr	Carriage return (P*)
change_scroll_region,	csr	cs	change to lines #1 through #2 (vt100) (PG)
clear_all_tabs,	tbc	ct	Clear all tab stops (P)
clear_screen,	clear	cl	Clear screen and home cursor (P*)
clr_eol,	el	ce	Clear to end of line (P)
clr_eos,	ed	cd	Clear to end of display (P*)
column_address,	hpa	ch	Set cursor column (PG)
command_character,	cmdch	CC	Term. settable cmd char in

			prototype
cursor_address,	cup	cm	Screen rel. cursor motion row #1 col #2 (PG)
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
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 (P*)
delete_line,	dl1	dl	Delete line (P*)
dis_status_line,	dsl	ds	Disable status line
down_half_line,	hd	hd	Half-line down (forward 1/2 linefeed)
enter_alt_charset_mode,	smacs	as	Start alternate character set (P)
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 stand out mode
enter_underline_mode,	smul	us	Start underscore mode
erase_chars	ech	ec	Erase #1 characters (PG)
exit_alt_charset_mode,	rmacs	ae	End alternate character set (P)
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 stand out mode
exit_underline_mode,	rmul	ue	End underscore mode
flash_screen,	flash	vb	Visible bell (may not move cursor)
form_feed,	ff	ff	Hardcopy terminal page eject (P*)
from_status_line,	fsl	fs	Return from status line
init_1string,	is1	i1	Terminal initialization string
init_2string,	is2	i2	Terminal initialization string
init_3string,	is3	i3	Terminal initialization string
init_file,	if	if	Name of file containing is
insert_character,	ich1	ic	Insert character (P)
insert_line,	il1	al	Add new blank line (P*)
insert_padding,	ip	ip	Insert pad after character inserted (p*)
key_backspace,	kbs	kb	Sent by backspace key
key_catab,	ktbc	ka	Sent by clear-all-tabs key
key_clear,	kclr	kC	Sent by clear screen or erase key
key_ctab,	kctab	kt	Sent by clear-tab key
key_dc,	kdch1	kD	Sent by delete character key
key_dl,	kdl1	kL	Sent by delete line key
key_down,	kcud1	kd	Sent by terminal down arrow key
key_eic,	krmir	kM	Sent by rmir or smir in insert mode
key_eol,	kel	kE	Sent by clear-to-end-of-line key
key_eos,	ked	kS	Sent by clear-to-end-of-screen key
key_f0,	kf0	k0	Sent by function key f0
key_f1,	kf1	k1	Sent by function key f1
key_f10,	kf10	ka	Sent by function key f10
key_f2,	kf2	k2	Sent by function key f2
key_f3,	kf3	k3	Sent by function key f3
key_f4,	kf4	k4	Sent by function key f4
key_f5,	kf5	k5	Sent by function key f5
key_f6,	kf6	k6	Sent by function key f6
key_f7,	kf7	k7	Sent by function key f7
key_f8,	kf8	k8	Sent by function key f8
key_f9,	kf9	k9	Sent by function key f9
key_home,	khome	kh	Sent by home key
key_ic,	kich1	kI	Sent by ins char/enter ins mode key
key_il,	kil1	kA	Sent by insert line
key_left,	kcub1	kl	Sent by terminal left arrow key

key_ll,	kll	kH	Sent by home-down key
key_npage,	knp	kN	Sent by next-page key
key_ppage,	kpp	kP	Sent by previous-page key
key_right,	kcuf1	kr	Sent by terminal right arrow key
key_sf,	kind	kF	Sent by scroll-forward/down key
key_sr,	kri	kR	Sent by scroll-backward/up key
key_stab,	khts	kT	Sent by set-tab key
key_up,	kcuu1	ku	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 funct key f0 if not f0
lab_f1,	lf1	l1	Labels on funct key f1 if not f1
lab_f10,	lf10	la	Labels on funct key f10 if not f10
lab_f2,	lf2	l2	Labels on funct key f2 if not f2
lab_f3,	lf3	l3	Labels on funct key f3 if not f3
lab_f4,	lf4	l4	Labels on funct key f4 if not f4
lab_f5,	lf5	l5	Labels on funct key f5 if not f5
lab_f6,	lf6	l6	Labels on funct key f6 if not f6
lab_f7,	lf7	l7	Labels on funct key f7 if not f7
lab_f8,	lf8	l8	Labels on funct key f8 if not f8
lab_f9,	lf9	l9	Labels on funct key f9 if not f9
meta_on,	smm	mm	Turn on meta mode (8th bit)
meta_off,	rmm	mo	Turn off meta mode
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 (PG*)
parm_delete_line,	dl	DL	Delete #1 lines (PG*)
parm_down_cursor,	cud	DO	Move cursor down #1 lines (PG*)
parm_ich,	ich	IC	Insert #1 blank chars (PG*)
parm_index,	indn	SF	Scroll forward #1 lines (PG)
parm_insert_line,	il	AL	Add #1 new blank lines (PG*)
parm_left_cursor,	cub	LE	Move cursor left #1 spaces (PG)
parm_right_cursor,	cuf	RI	Move cursor right #1 spaces (PG*)
parm_rindex,	rin	SR	Scroll backward #1 lines (PG)
parm_up_cursor,	cuu	UP	Move cursor up #1 lines (PG*)
pkey_key,	pfkey	pk	Prog funct key #1 to type string #2
pkey_local,	pfloc	pl	Prog funct key #1 to execute

			string #2
pkey_xmit,	pfx	px	Prog funct key #1 to xmit string #2
print_screen,	mc0	ps	Print contents of the screen
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. (PG*)
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 (set row) (PG)
save_cursor,	sc	sc	Save cursor position (P)
scroll_forward,	ind	sf	Scroll text up (P)
scroll_reverse,	ri	sr	Scroll text down (P)
set_attributes,	sgr	sa	Define the video attributes (PG9)
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
tab,	ht	ta	Tab to next 8 space hardware tab stop
to_status_line,	tsl	ts	Go to status line, column #1
underline_char,	uc	uc	Underscore one char and move past it
up_half_line,	hu	hu	Half-line up (reverse 1/2 linefeed)
init_prog,	iprog	iP	Path name of program for init
key_a1,	ka1	K1	Upper left of keypad
key_a3,	ka3	K3	Upper right of keypad
key_b2,	kb2	K2	Center of keypad
key_c1,	kc1	K4	Lower left of keypad
key_c3,	kc3	K5	Lower right of keypad

prtr_non,	mc5p	pO	Turn on the printer for #1 bytes

A Sample Entry

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

```
concept100|c100|concept|c104|c100-4p|concept 100,
    am, bel=^G, blank=\EH, blink=\EC, clear=^L$<2*>, cnorm=\Ew,
    cols#80, cr=^M$<9>, cub1=^H, cud1=^J, cuf1=\E=,
    cup=\Ea%p1%' '%+%c%p2%' '%+%c,
    cuu1=\E;, cvvis=\EW, db, dch1=\E^A$<16*>, dim=\EE, dl1=\E^B$<3*>,
    ed=\E^C$<16*>, el=\E^U$<16>, eo, flash=\Ek$<20>\EK, ht=\t$<8>,
    il1=\E^R$<3*>, in, ind=^J, .ind=^J$<9>, ip=$<16*>,
    is2=\EU\Ef\E7\E5\E8\El\ENH\EK\E\200\Eo&\200\Eo\47\E,
    kbs=^h, kcub1=\E>, kcud1=\E<, kcuf1=\E=, kcuu1=\E;,
    kf1=\E5, kf2=\E6, kf3=\E7, khome=\E?,
    lines#24, mir, pb#9600, prot=\EI, rep=\Er%p1%c%p2%' '%+%c$<.2*>,
    rev=\ED, rmcup=\Ev    $<6>\Ep\r\n, rmir=\E\200, rmkx=\Ex,
    rmso=\Ed\Ee, rmul=\Eg, rmul=\Eg, sgr0=\EN\200,
    smcup=\EU\Ev  8p\Ep\r, smir=\E^P, smkx=\EX, smso=\EE\ED,
    smul=\EG, tabs, ul, vt#8, xenl,
```

Entries may continue onto multiple lines by placing white space at the beginning of each line except the first. Comments may be included on lines beginning with "#". 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 the size of particular delays, 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* (i.e., 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.

Finally, string valued capabilities, such as `el` (clear to end of line sequence) are given by the two-character code, an "=", and then a string ending at the next following ",". 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` to provide this delay. The delay can be either a number, for example, "20," or a number followed by an "*", like "3*". 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 `xenl` 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 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 \s` give a newline, linefeed, return, tab, backspace, formfeed, and space. Other escapes include \^ for ^, \\ for \, \, for comma, \: for :, and \0 for null. (\0 will produce \200, which does not terminate a string but behaves as a null character on most terminals.) Finally, characters may be given as three octal digits after a `\`.

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.

Preparing Descriptions

We now outline how to prepare descriptions of terminals. 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` to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the `terminfo` file to describe it or bugs in `vi`. To test a new terminal description easily, you may 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 right (if the terminal manufacturer did not document it), a severe test is to edit `/etc/passwd` at 9600 baud, delete 16 or so lines from the middle of the screen, then hit the "u" key several times quickly. If the terminal messes up, 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 is a CRT 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 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=" 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 CRT terminal. Programs should never attempt to backspace around the left edge, unless bw is given, and 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; i.e., 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 carriage return or linefeed, it may still be possible to craft a working nel out of one or both of them.

These capabilities suffice to describe hardcopy and "glass-tty" 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 | 3 | 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. 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.

The % encodings have the following meanings:

```
%%      outputs '%'
%d      print pop() as in printf
%2d     print pop() like %2d
%3d     print pop() like %3d
%02d
%03d    as in printf
%c      print pop() gives %c
%s      print pop() gives %s

%p[1-9] push ith parm
```

```
%P[a-z] set variable [a-z] to pop()
%g[a-z] get variable [a-z] and push it
%'c'    char constant c
%{nn}   integer constant nn

%+ %- %* %/ %m
        arithmetic (%m is mod): push(pop() op pop())
%& %| %^    bit operations: push(pop() op pop())
%= %> %<    logical operations: push(pop() op pop())
%! %~   unary operations push(op pop())
%i      add 1 to first two parms (for ANSI terminals)

%? expr %t thenpart %e elsepart %;
     if-then-else, %e elsepart is optional.
     else-if's are possible ala Algol 68:
     %? c1 %t b1 %e c2 %t b2 %e c3 %t b3 %e c4 %t b4 %e %;
     ci are conditions, bi are bodies.
```

Binary operations are in postfix form with the operands in the usual order. That is, to get x-5 one would use "`gx%{5}%-`".

Consider the HP2645, which, to get to row 3 and column 12, needs to be sent a `\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 printed as two digits. Thus its `cup` capability is "`cup=6\E&%p2%2dc%p1%2dY`".

The Microterm 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, thus "`cup=\E=%p1%' '%+%c%p2%' '%+%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.

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 hp2645) and can be used in preference to `cup`. If there are parameterized local motions (e.g., 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.

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 HP terminals cannot be used for `home`.)

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 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 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.

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 may bring nonblank lines up from below or that scrolling back with `ri` may bring down nonblank lines.

Insert/Delete Character

There are two basic kinds of intelligent terminals with respect to insert/delete character 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 vs. 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 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.

It is occasionally necessary to move around while in insert mode to delete characters on the same line (e.g., 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", 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.) 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 Microterm 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 there is a sequence to set arbitrary combinations of modes, this should be given as `sgr` (set attributes), taking 9 parameters. Each parameter is either 0 or 1, as the corresponding attribute is on or off. The 9 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.

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 HP 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.

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`. 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 both 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 your terminal correctly generates underlined characters (with no special codes needed) even though it does not overstrike, then you should give the capability `ul`. If overstrikes are erasable with a blank, then this should be indicated by giving `eo`.

Keypad

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 HP 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, ..., f10, the codes they send can be given as `kf0`, `kf1`, ..., `kf10`. If these 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.

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 "back-tab" 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 the `tset` command 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.

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 are normally sent to the terminal, by the `tset` program, each time the user logs in. They will be printed in the following order: `is1`; `is2`; setting tabs using `tbc` and `hts`; `if`; running the program `iprog`; and finally `is3`. 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`. A pair of sequences that does a harder reset from a totally unknown state can be analogously given as `rs1`, `rs2`, `rf`, and `rs3`, analogous to `is2` and `if`. These strings are output by the `reset` program, which is used when the terminal gets into a wedged state. Commands are normally placed in `rs2` and `rf` only if they produce annoying effects on the screen and are not necessary when logging in. For example, the command to set the vt100 into 80-column mode would normally be part of `is2`, but it causes an annoying glitch of the screen and is not normally needed since the terminal is usually already in 80 column mode.

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). If a more complex sequence is needed to set the tabs than can be described by this, the sequence can be placed in `is2` or `if`.

Delays

Certain capabilities control padding in the teletype driver. These are primarily needed by hard copy terminals, and are used by the *tset* program to set teletype modes appropriately. Delays embedded in the capabilities `cr`, `ind`, `cub1`, `ff`, and `tab` will cause the appropriate delay bits to be set in the teletype driver. If `pb` (padding baud rate) is given, these values can be ignored at baud rates below the value of `pb`.

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 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 to go to the beginning of the status line and to 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 parameter `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, e.g., `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`.

If the terminal can move up or down half a line, this can be indicated with `hu` (half-line up) and `hd` (half-line down). This is primarily useful for superscripts and subscripts on hardcopy terminals. If a hardcopy terminal can eject to the next page (form feed), give this as `ff` (usually CONTROL-L).

If there is a command to repeat a given character a given number of times (to save time transmitting a large number of identical characters) this can be indicated with the parameterized string `rep`. The first parameter is the character to be repeated and the second is the number of times to repeat it. Thus,

```
tparm(repeat_char, 'x', 10)
```

is the same as "`xxxxxxxxxx`".

If the terminal has a settable command character, such as the TEKTRONIX 4025, this can be indicated with `cmdch`. A prototype command character is chosen which is used in all capabilities. This character is given in the `cmdch` capability to identify it. The following convention is supported on some UNIX systems: The environment is to be searched for a `CC` variable, and if found, all occurrences of the prototype character are replaced with the character in the environment variable.

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.

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.

If the terminal is one of those supported by the UNIX virtual terminal protocol, the terminal number can be given as `vt`.

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. It is undefined whether the text is also displayed on the terminal screen when the printer is on. 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. All text, including `mc4`, is transparently passed to the printer while an `mc5p` is in effect.

Strings to program function keys can be given as `pfkey`, `pfloc`, and `pfx`. Each of these strings takes two parameters: the function key number to program (from 0 to 10) and the string to program it with. Function key numbers out of this range may program undefined keys in a terminal dependent manner. The difference between the capabilities is that `pfkey` causes pressing the given key to be the same as the user typing the given string; `pfloc` causes the string to be executed by the terminal in local; and `pfx` causes the string to be transmitted to the computer.

BUGS

Hazeltine terminals, which do not allow tilde characters to be displayed should indicate `hz`.

Terminals which ignore a linefeed immediately after an `am` wrap, such as the Concept and vt100, should indicate `xenl`.

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

Teleray terminals, where tabs turn all characters moved over to blanks, should indicate `xt` (destructive tabs). This glitch is also taken to mean that it is not possible to position the cursor on top of a ''magic cookie'', that to erase standout mode it is instead necessary to use delete and insert line.

The Beehive Superbee, which is unable to correctly transmit the escape or control-C characters, has `xsb`, indicating that the f1 key is used for ESCAPE and f2 for CONTROL-C. (Only certain Superbees have this problem, depending on the ROM.)

Other specific terminal problems may be corrected by adding more capabilities of the form `xx`.

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 cancelled by placing `xx@` to the left of the capability definition, where xx is the capability. For example, the entry

```
2621-nl, smkx@, rmkx@, use=2621,
```

defines a 2621-nl that does not have the `smkx` or `rmkx` capabilities, and hence does not turn on the function key labels when in visual mode. This is useful for different modes for a terminal, or for different user preferences.

FILES

`/usr/lib/terminfo/?/*`	files containing binary terminal descriptions

SEE ALSO

`tic`(1M), `curses`(3X), `printf`(3S), `term`(4).

NAME

ttytype — database of terminal types by port

DESCRIPTION

ttytype is a database containing, for each tty port on the system, the kind of terminal that is attached to it. There is one line per port, containing the terminal kind (as a name listed in termcap(4)), a space, and the name of the tty, minus /dev/.

This information is read by tset(1) and by login(1) to initialize the TERM environment variable at login time.

EXAMPLES

```
dw console
3a tty0
h19 tty1
h19 tty2
du ttyd0
```

FILES

/etc/ttytype

SEE ALSO

tset(1), login(1).

NAME

tzfile — time-zone information

SYNOPSIS

`#include <tzfile.h>`

DESCRIPTION

The time-zone information files used by tzset(3) begin with bytes reserved for future use, followed by four 4-byte values of type long, written in a standard byte order where the high-order byte of the value is written first. These values are, in order:

tzh_ttisstdcnt
: The number of standard/wall indicators stored in the file

tzh_leapcnt
: The number of leap seconds for which data is stored in the file

tzh_timecnt
: The number of "transition times" for which data is stored in the file

tzh_typecnt
: The number of "local time types" for which data is stored in the file (must not be 0)

tzh_charcnt
: The number of characters of ''time-zone abbreviation strings'' stored in the file

The above header is followed by tzh_timecnt 4-byte values of type long, sorted in ascending order. These values are written in standard byte order. Each is used as a transition time (as returned by time(2)) where the rules for computing local time change. Next come tzh_timecnt 1-byte values of type unsigned char. Each one tells which of the different types of ''local time'' types described in the file is associated with the same-indexed transition time. These values serve as indices into an array of ttinfo structures that appears next in the file. These structures are defined as follows:

```
struct ttinfo {
      long       tt_gmtoff;
      int        tt_isdst;
      unsigned inttt_abbrind;
};
```

Each structure is written as a 4-byte value for `tt_gmtoff` of type `long`, in a standard byte order, followed by a 1-byte value for `tt_isdst` and a 1-byte value for `tt_abbrind`. In each structure, `tt_gmtoff` gives the number of seconds to be added to Greenwich mean time (GMT), `tt_isdst` tells whether `tm_isdst` should be set by `localtime`(3), and `tt_abbrind` serves as an index into the array of time-zone abbreviation characters that follow the `ttinfo` structure(s) in the file.

Then there are `tzh_leapcnt` pairs of 4-byte values, written in a standard byte order. The first value of each pair gives the time (as returned by `time`(2)) at which a leap second occurs; the second gives the `total` number of leap seconds to be applied after the given time. The pairs of values are sorted in ascending order by time.

Finally, there are `tzh_ttisstdcnt` standard/wall indicators, each stored as a 1-byte value. They tell whether the transition times associated with local time types are specified as standard time or wall-clock time and are used when a time-zone file is used in handling POSIX-style time-zone environment variables.

`localtime` uses the first standard-time `ttinfo` structure in the file (or simply the first `ttinfo` structure in the absence of a standard-time structure) if either `tzh_timecnt` is 0 or the time argument is less than the first transition time recorded in the file.

SEE ALSO

`ctime`(3), `tzic`(1M), `tzdump`(1M).

NAME

ufs — format of a UFS file-system volume

SYNOPSIS

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

DESCRIPTION

Every Berkeley 4.2 file-system (UFS) storage volume (for example, a hard disk or a floppy disk) has a common format for certain vital information. Each volume is divided into a certain number of blocks. The block size is a parameter of the file system. Sectors 0 to 7 on a file system may be used to contain bootstrap programs. The first **superblock** for the file system is located at sector 8.

The actual file system begins at sector 16 with the first alternate superblock. The layout of the superblock as defined by the include file <ufs/fs.h> is:

```
#define  FS_MAGIC 0x011954
struct   fs {
         struct   fs *fs_link;
                  /* linked list of file systems */
         struct   fs *fs_rlink;
                  /*     used for incore superblocks */
         daddr_t  fs_sblkno;
                  /* addr of superblock in filesys */
         daddr_t  fs_cblkno;
                  /* offset of cyl-block in filesys */
         daddr_t  fs_iblkno;
                  /* offset of inode-blocks in filesys */
         daddr_t  fs_dblkno;
                  /* offset of first data after cg */
         long     fs_cgoffset;
                  /* cylinder group offset in cylinder */
         long     fs_cgmask;
                  /* used to calc mod fs_ntrak */
         time_t   fs_time;
                  /* last time written */
         long     fs_size;
                  /* number of blocks in fs */
```

```
        long    fs_dsize;
                /* number of data blocks in fs */
        long    fs_ncg;
                        /* number of cylinder groups */
        long    fs_bsize;
                /* size of basic blocks in fs */
        long    fs_fsize;
                /* size of frag blocks in fs */
        long    fs_frag;
                /* number of frags in a block in fs */
/* these are configuration parameters */
        long    fs_minfree;
                /* minimum percentage of free blocks */
        long    fs_rotdelay;
                /* num of ms for optimal next block */
        long    fs_rps;
                        /* disk revolutions per second */
/* these fields can be computed from the others */
        long    fs_bmask;
                /* ``blkoff'' calc of blk offsets */
        long    fs_fmask;
                /* ``fragoff'' calc of frag offsets */
        long    fs_bshift;
                /* ``lblkno'' calc of logical blkno */
        long    fs_fshift;
                /* ``numfrags'' calc number of frags */
/* these are configuration parameters */
        long    fs_maxcontig;
                /* max number of contiguous blks */
        long    fs_maxbpg;
                /* max number of blks per cyl group */
/* these fields can be computed from the others */
        long    fs_fragshift;
                /* block to frag shift */
        long    fs_fsbtodb;
                /* fsbtodb and dbtofsb shift constant */
        long    fs_sbsize;
                /* actual size of superblock */
        long    fs_csmask;
                /* csum block offset */
        long    fs_csshift;
                /* csum block number */
```

```
        long    fs_nindir;
                /* value of NINDIR */
        long    fs_inopb;
                /* value of INOPB */
        long    fs_nspf;
                /* value of NSPF */
        long    fs_optim;
                /* optimization preference */
        long    fs_sparecon[2];
                /* reserved for future constants */
        long    fs_state;
                /* file-system state */
        long    fs_id[2];
                /* file-system id */
/* sizes determined by number of cylinder groups and their sizes */
        daddr_t fs_csaddr;
                /* blk addr of cyl grp summary area */
        long    fs_cssize;
                /* size of cyl grp summary area */
        long    fs_cgsize;
                /* cylinder group size */
/* these fields should be derived from the hardware */
        long    fs_ntrak;
                /* tracks per cylinder */
        long    fs_nsect;
                /* sectors per track */
        long    fs_spc;
                /* sectors per cylinder */
/* this comes from the disk driver partitioning */
        long    fs_ncyl;
                /* cylinders in file system */
/* these fields can be computed from the others */
        long    fs_cpg;
                /* cylinders per group */
        long    fs_ipg;
                /* inodes per group */
        long    fs_fpg;
                /* blocks per group * fs_frag */
/* this data must be recomputed after crashes */
        struct  csum fs_cstotal;
                /* cylinder summary information */
/* these fields are cleared at mount time */
```

```
        char    fs_fmod;
                /* superblock modified flag */
        char    fs_clean;
                /* file system is clean flag */
        char    fs_ronly;
                /* mounted read-only flag */
        char    fs_flags;
                /* currently unused flag */
        char    fs_fsmnt[MAXMNTLEN];
                /* name mounted on */
        char    fs_fsname[6];
                /* file-system name */
        char    fs_fpack[6];
                /* file-system pack name */
/* these fields retain the current block allocation info */
        long    fs_cgrotor;
                /* last cg searched */
        struct  csum *fs_csp[MAXCSBUFS];
                /* list of fs_cs info buffers */
        long    fs_cpc;
                /* cyl per cycle in postbl */
        short   fs_postbl[MAXCPG][NRPOS];
                /* head of blocks for each rotation */
        long    fs_magic;
                /* magic number */
        u_char  fs_rotbl[1];
                /* list of blocks for each rotation */
/* actually longer */
};
```

A disk may contain one or more partitions. A disk partition may contain at most one file system. A file system consists of a number of cylinder groups. Each cylinder group has inodes and data.

A BSD file system is described by its superblock, which in turn describes the cylinder groups. The superblock is critical data and is replicated in each cylinder group to protect against catastrophic loss. This is done at file-system creation time. In addition, the critical superblock data does not change, so the copies need not be referenced further unless disaster strikes.

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

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

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

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

fs_minfree gives the minimum acceptable percentage of file system blocks that may be free. If the free list drops below this level, only the superuser may continue to allocate blocks. This may be set to 0 if no reserve of free blocks is deemed necessary; however, severe performance degradations occur if the file-system is run at greater than 90% full. Thus the default value of fs_minfree is 10%.

Empirically, the best trade-off between block fragmentation and overall disk utilization at a loading of 90% comes with a fragmentation of 4; thus the default fragment size is a fourth of the block size.

Cylinder-group Related Limits

Each cylinder keeps track of the availability of blocks at different rotational positions so that sequential blocks can be laid out with minimum rotational latency. NRPOS is the number of rotational positions that are distinguished. With NRPOS 8 the resolution of the summary information is 2 ms for a typical 3600 rpm drive.

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

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

MAXIPG bounds the number of inodes per cylinder group and is needed only to keep the structure simpler by having the only a single variable size element (the free bit map). Note that MAXIPG must be a multiple of INOPB(fs).

MINBSIZE is the smallest allowable block size. With MINBSIZE of 4096, it is possible to create files of size 2^32 with only two levels of indirection. MINBSIZE must be big enough to hold a cylinder group block, so changes to (struct cg) must keep its size within MINBSIZE. MAXCPG is limited only to dimension an array in (struct cg); it can be made larger as long as that structure's size remains within the bounds dictated by MINBSIZE. Note that superblocks are never more than size SBSIZE.

The pathname on which the file system is mounted is maintained in fs_fsmnt. MAXMNTLEN defines the amount of space allocated in the superblock for this name. The limit on the amount of summary information per file system is defined by MAXCSBUFS. It is currently parameterized for a maximum of two million cylinders.

Per cylinder-group information is summarized in blocks allocated from the data blocks of the first cylinder. These blocks are read in, from the location indicated by fs_csaddr, in addition to the superblock. The size of the summary information is given by fs_cssize.

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

Superblock for a File System

MAXBPC bounds the size of the rotational layout tables and is limited by the fact that the superblock is of size SBSIZE. The size of these tables is inversely proportional to the block size of the file system. The size of the tables is increased when sector sizes are not powers of two, as this increases the number of cylinders included before the rotational pattern repeats (`fs_cpc`). The size of the rotational layout tables is derived from the number of bytes remaining in (`struct fs`) .

`MAXBPG` bounds the number of blocks of data per cylinder group and is limited by the fact that cylinder groups are at most one block. The size of the free-block table is derived from the size of blocks and the number of remaining bytes in the cylinder group structure (`struct cg`).

Inode

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

SEE ALSO

`newfs`(1M), `svfs`(4).

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 */
     char     ut_host[16];         /* host name if remote */
};

/*  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

/usr/include/utmp.h
/etc/utmp
/etc/wtmp

SEE ALSO

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

NAME

ypfiles — the Yellow Pages database and directory structure

DESCRIPTION

The yellow pages (YP) network lookup service uses a database of dbm files in the directory hierarchy at /etc/yp. A dbm database consists of two files, created by calls to the dbm(3X) library package. One has the filename extension .pag and the other has the filename extension .dir. For instance, the database named hst.nm, is implemented by a pair of files, hst.nm.pag and hst.nm.dir. A dbm database served by the YP is called a YP *map*. A YP *domain* is a named set of YP maps. Each YP domain is implemented as a subdirectory of /etc/yp containing the map. Any number of YP domains can exist. Each may contain any number of maps.

No maps are required by the YP lookup service itself, although they may be required for the normal operation of other parts of the system. There is no list of maps which YP serves; if the map exists in a given domain and a client asks about it, the YP will serve it. For a map to be accessible consistently, it must exist on all YP servers that serve the domain. To provide data consistency between the replicated maps, an entry to run ypxfr periodically should be made in /usr/lib/crontab on each server. More information on this topic is in ypxfr(1M).

YP maps should contain two distinguished key-value pairs. The first is the key YP_LAST_MODIFIED, having as a value a ten-character ASCII order number. The order number should be the UNIX time in seconds when the map was built. The second key is YP_MASTER_NAME, with the name of the YP master server as a value. makedbm generates both key-value pairs automatically. A map that does not contain both key-value pairs can be served by the YP, but the ypserv process will not be able to return values for "Get order number" or "Get master name" requests. In addition, values of these two keys are used by ypxfr when it transfers a map from a master YP server to a slave. If ypxfr cannot figure out where to get the map or if it is unable to determine whether the local copy is more recent than the copy at the master, you must set extra command line switches when you run it.

YP maps must be generated and modified only at the master server. They are copied to the slaves using ypxfr(1M) to avoid potential byte-ordering problems among YP servers running on machines with different architectures, and to minimize the amount of disk space required for the dbm files. The YP database can be initially set up for both masters and slaves by using ypinit(1M).

After the server databases are set up, it is probable that the contents of some maps will change. In general, some ASCII source version of the database exists on the master, and it is changed with a standard text editor. The update is incorporated into the YP map and is propagated from the master to the slaves by running /etc/yp/Makefile. All vendor-supplied maps have entries in /etc/yp/Makefile; if you add a YP map, edit the this file to support the new map. The makefile uses makedbm to generate the YP map on the master, and yppush to propagate the changed map to the slaves. yppush is a client of the map ypservers, which lists all the YP servers. For more information on this topic, see yppush(1M).

SEE ALSO

makedbm(1M), ypinit(1M), ypmake(1M), ypxfr(1M), yppush(1M), yppoll(1M), ypserv(1M), rpcinfo(1M), *A/UX Network Applications Programming*, Appendix E: *YP Protcol Specification.*

Table of Contents

Section 5: Miscellaneous Facilities

intro(5) .... introduction to miscellaneous facilities
ae(5) .... 3Com 10 Mb/s Ethernet interface
arp(5P) .... Address Resolution Protocol
ascii(5) .... map of ASCII character set
environ(5) .... user environment
eqnchar(5) .... special character definitions for eqn and neqn
fcntl(5) .... file control options
font(5) .... description files for device-independent troff
greek(5) .... graphics for the extended TTY-37 type-box
icmp(5P) .... Internet Control Message Protocol
inet(5F) .... Internet protocol family
ip(5P) .... Internet Protocol
lo(5) .... software loopback network interface
man(5) .... macros for formatting entries in this manual
math(5) .... math functions and constants
me(5) .... macros for formatting papers
mm(5) .... macro package for formatting documents
mptx(5) .... the macro package for formatting a permuted index
ms(5) .... text formatting macros
mv(5) .... a troff macro package for typesetting viewgraphs and slides
nterm(5) .... terminal driving tables for nroff
prof(5) .... profile within a function
regexp(5) .... regular expression compile and match routines
stat(5) .... data returned by stat system call
tcp(5P) .... Internet Transmission Control Protocol
term(5) .... conventional names for terminals
troff(5) .... description of troff output language
types(5) .... primitive system data types
udp(5P) .... Internet User Datagram Protocol
values(5) .... machine-dependent values

NAME

intro — introduction to miscellaneous facilities

SYNOPSIS

```
#include <sys/socket.h>
#include <net/route.h>
#include <net/if.h>
```

DESCRIPTION

This section describes miscellaneous facilities (such as macro packages, character set tables, and so forth) and networking facilities (such as network protocols) available in the system.

Macro packages, character set tables and hardware support for network interfaces are found among the standard Section 5 entries. Entries describing a protocol family are marked 5F, while entries describing protocol use are marked 5P.

NETWORKING FACILITIES

All network protocols are associated with a specific protocol family. 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 may 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`(2N) type. It is not required that a protocol family support all socket types. A protocol family may contain multiple protocols supporting the same socket abstraction.

A protocol supports one of the socket abstractions detailed in `socket`(2N). A specific protocol may be accessed either by creating a socket of the appropriate type and protocol family, or by requesting the protocol explicitly when creating a socket. Protocols normally accept only one type of address format, usually determined by the addressing structure inherent in the design of the protocol family/network architecture. Certain semantics of the basic socket abstractions are protocol specific. All protocols are expected to support the basic model for their particular socket type, but may, in addition, provide nonstandard facilities or extensions to a mechanism. For example, a protocol supporting the `SOCK_STREAM` abstraction may allow more than one byte of out-of-band data to be transmitted per out-of-band message.

A network interface is similar to a device interface. Network interfaces comprise the lowest layer of the networking subsystem, interacting with the actual transport hardware. An interface may support one or more protocol families or address formats.

PROTOCOLS

The system currently supports only the DARPA Internet protocols fully. Raw socket interfaces are provided to IP protocol layer of the DARPA Internet, to the IMP link layer (1822), and to Xerox PUP-1 layer operating on top of 3Mb/s Ethernet interfaces. Consult the appropriate manual pages in this section for more information regarding the support for each protocol family.

ADDRESSING

Associated with each protocol family is an address format. The following address formats are used by the system:

```
#define AF_UNIX     1   /*local to host (pipes, portals)*/
#define AF_INET     2   /*internetwork: UDP, TCP, etc.*/
#define AF_IMPLINK  3   /*arpanet imp addresses*/
#define AF_PUP      4   /*pup protocols: e.g. BSP*/
```

Note: Only `AF_INET` is appropriate for this implementation.

ROUTING

The network facilities provided 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. Routing table manipulations may only be carried out by superuser.

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;
};
```

with `rt_flags` defined from,

```
#define RTF_UP       0x1   /*route usable*/
#define RTF_GATEWAY  0x2   /*destination is a gateway*/
#define RTF_HOST     0x4   /*host entry (net otherwise)*/
```

Routing table entries come in three flavors: for a specific host, for all hosts on a specific network, for any destination not matched by entries of the first two types (a wildcard route). When the system is booted, each network interface autoconfigured installs a routing table entry when it wishes to have packets sent through it. Normally the interface specifies the route through it is a ''direct'' connection to the destination host or network. If the route is direct, the transport layer of a protocol family usually requests the packet be sent to the same host specified in the packet. Otherwise, the interface may be requested to address the packet to an entity different from the eventual recipient (that is, the packet is forwarded).

Routing table entries installed by a user process may not specify the hash, reference count, use, or interface fields; these are filled in by the routing routines. If a route is in use when it is deleted (`rt_refcnt` is nonzero), the resources associated with it will not be reclaimed until further references to it are released.

The routing code returns `EEXIST` if requested to duplicate an existing entry, `ESRCH` if requested to delete a nonexistent entry, or `ENOBUFS` 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. This value is used to select among multiple routes to the same destination. When multiple routes to the same destination exist, the least-used route is selected.

A wildcard routing entry is specified with a zero destination address value. Wildcard routes are used only when the system fails to find a route to the destination host and network. The combination of wildcard routes and routing redirects can provide an economical mechanism for routing traffic.

INTERFACES

Each network interface in a system corresponds to a path through which messages may be sent and received. A network interface usually has a hardware device associated with it, though certain interfaces such as the loopback interface, `lo`(5), do not.

At boot time, each interface which has underlying hardware support makes itself known to the system during the autoconfiguration process. Once the interface has acquired its address, it is expected to install a routing table entry so that messages may be routed through it. Most interfaces require some part of their address specified with an `SIOCSIFADDR` `ioctl` before they will allow traffic to flow through them. On interfaces where the network-link layer address mapping is static, only the network number is taken from the ioctl; the remainder is found in a hardware specific manner. On interfaces which provide dynamic network-link layer address mapping facilities (for example, 10Mb/s Ethernets), the entire address specified in the ioctl is used.

The following `ioctl` calls may be used to manipulate network interfaces. Unless specified otherwise, the request takes an `ifrequest` structure as its parameter. This structure has the form

```
#define ifr_addr    ifr_ifru.ifru_addr    /* address */
#define ifr_dstaddr ifr_ifru.ifru_dstaddr /* other end of
                                             p-to-p link */
#define ifr_flags   ifr_ifru.ifru_flags   /* flags */

struct ifreq {
       char    ifr_name[16];     /* name of interface
                                    (e.g. "ec0") */
       union {
               struct    sockaddr ifru_addr;
               struct    sockaddr ifru_dstaddr;
               short     ifru_flags;
       } ifr_ifru;
};
```

`SIOCSIFADDR`	Set interface address. Following the address assignment, the "initialization" routine for the interface is called.
`SIOCGIFADDR`	Get interface address.
`SIOCSIFDSTADDR`	Set point to point address for interface.
`SIOCGIFDSTADDR`	Get point to point address for interface.

SIOCSIFFLAGS	Set interface flags field. If the interface is marked down, any processes currently routing packets through the interface are notified.
SIOCGIFFLAGS	Get interface flags.
SIOCGIFCONF	Get interface configuration list. This request takes an `ifconf` structure (see below) as a value-result parameter. The `ifc_len` field should be initially set to the size of the buffer pointed to by `ifc_buf`. On return it will contain the length, in bytes, of the configuration list.

```
/*
 * Structure used in SIOCGIFCONF request.
 * Used to retrieve interface configuration
 * for machine (useful for programs which
 * must know all networks accessible).
 */
#define ifc_buf ifc_ifcu.ifcu_buf  /* buffer address */
#define ifc_req ifc_ifcu.ifcu_req  /* array of structures
                                      returned */
struct  ifconf {
        int     ifc_len;           /* size of associated
                                      buffer */
        union {
                caddr_t ifcu_buf;
                struct  ifreq *ifcu_req;
        } ifc_ifcu;
};
```

SEE ALSO

routed(1M), socket(2N), ioctl(2).

NAME

ae — 3Com 10 Mb/s Ethernet interface

DESCRIPTION

The ae interface provides host access to an industry standard 10 Mb/s Ethernet.

The host's Internet address is specified at boot time with an SIOCSIFADDR ioctl. The hosts's Ethernet address is read from ROM on the Ethernet board using etheraddr(1M). The ae interface employs the address resolution protocol described in arp(5P) to dynamically map between Internet and Ethernet addresses on the local network.

DIAGNOSTICS

```
ae%d: init failed
```

The NIC chip on the Ethernet board would not initalize.

```
ae%d transmitter frozen - resetting
```

A packet transmission failed to complete within a predetermined timeout period.

```
ae%d spurious interrupt
```

An interrupt was received but no operation was active.

```
ae%d: can't handle af%d
```

The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

SEE ALSO

etheraddr(1M), inet(5F), intro(5), arp(5P).

FILES

/etc/boot.d/ae6
/etc/master.d/ae6
/etc/startup.d/ae6

NAME

arp — Address Resolution Protocol

DESCRIPTION

arp is a protocol used to dynamically map between DARPA Internet and 10Mb/s Ethernet addresses on a local area network. It is used by all the 10Mb/s Ethernet interface drivers and is not directly accessible to users.

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 messages are transmitted. arp itself is not Internet or Ethernet specific; this implementation, however, is. arp will queue at most one packet while waiting for a mapping request to be responded to; only the most recently "transmitted" packet is kept.

arp watches passively for hosts impersonating the local host (i.e. a host which responds to an arp mapping request for the local host's address) and will, optionally, periodically probe a network looking for impostors.

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

`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. It contains:

```
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

SYNOPSIS

```
extern char **environ;
```

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:

PATH — The sequence of directory prefixes that sh, time, nice(1), etc., apply in searching for a file known by an incomplete path name. The prefixes are separated by :. login(1) sets

PATH=:/bin:/usr/bin

HOME — A user's login directory, set by login(1) from the password file passwd(4).

TERM — The kind of terminal for which output is to be prepared. This information is used by commands, such as nroff, more, or vi, which may exploit special terminal capabilities. See /etc/termcap or (termcap(4)) for a list of terminal types.

SHELL — The file name of the user's login shell.

TERMCAP — The string describing the terminal in TERM, or the name of the termcap file, see termcap(4).

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

LOGNAME — The login name of the user.

TZ — Time zone information. The format is xxx*n*zzz where xxx is standard local time zone abbreviation, *n* is the difference is hours from GMT, and zzz is the abbreviation for the daylight-saving local time zone, if any; for example, EST5EDT.

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

SEE ALSO

csh(1), ex(1), ksh(1), login(1), sh(1), exec(2), system(3S), termcap(4), tty(7).

NAME

eqnchar — special character definitions for eqn and neqn

SYNOPSIS

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

eqn /usr/pub/cateqnchar [*options*] [–]*files*] | troff [*options*]

neqn /usr/pub/eqnchar [*options*] [–]*files*] | troff [*options*]

eqn -Taps /usr/pub/apseqnchar [*options*] [–]*files*] | troff [*options*]

DESCRIPTION

/usr/pub/eqnchar contains troff(1) and nroff(1) character definitions for constructing characters that are not ordinarily available on a phototypesetter or printer. These definitions are primarily intended for use with eqn(1) and neqn(1).

For a complete list of input and output characters contained in /usr/pub/eqnchar, see the "eqn Reference" in *A/UX Text Processing Tools.*

/usr/pub/apseqnchar is a version of eqnchar tailored for the Autologic APS-5 phototypesetter. If you use apseqnchar, output will not look optimal on other phototypesetters. cateqnchar is more "device independent," and should look reasonable on any device supported by troff(1). You may link /usr/pub/eqnchar to /usr/pub/cateqnchar or to /usr/pub/apseqnchar. By default, /usr/pub/eqnchar is linked to /usr/pub/apseqnchar.

FILES

/usr/pub/eqnchar
/usr/pub/apseqnchar
/usr/pub/cateqnchar

SEE ALSO

eqn(1), neqn(1), troff(1).
"eqn Reference" in *A/UX Text Processing Tools.*

NAME

fcntl — file control options

SYNOPSIS

#include <fcntl.h>

DESCRIPTION

fcntl(2) provides for control over open files. The include file describes requests and arguments to fcntl(2) and open(2).

```
#ifndef __fcntl_h
#define __fcntl_h

/* POSIX requires types; most applications don't do this yet! */
#ifndef __sys_types_h
#include <sys/types.h>
#endif /* !__sys_types_h */

/* Flag values accessible to open(2) and fcntl(2) */
/*  (The first three can only be set by open) */
#if defined(_SYSV_SOURCE) || defined(_POSIX_SOURCE)
#ifndef __sys_file_h
#define O_RDONLY      0
#define O_WRONLY      1
#define O_RDWR  2
#define O_APPEND      010     /* append (writes
                                 guaranteed at the end) */

/* 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 */

/* file segment locking set data type - information passed */
/* to system by user */
struct flock {
        short   l_type;
        short   l_whence;
        long    l_start;
        long    l_len;  /* len = 0 means until end of file */
```

```
        int     l_pid;
};

/* file segment locking types */
#define F_RDLCK 01      /* Read lock */
#define F_WRLCK 02      /* Write lock */
#define F_UNLCK 03      /* Remove locks */

#endif /* _SYSV_SOURCE || _POSIX_SOURCE */

#ifdef _BSD_SOURCE
/* Additional fcntl(2) request */
#define F_GETOWN        8       /* Get owner */
#define F_SETOWN        9       /* Set owner */
#endif /* _BSD_SOURCE */

#ifdef _POSIX_SOURCE
/* File access mode mask */
#define O_ACCMODE       03

/* POSIX-defined argument to F_SETFD */
#define FD_CLOEXEC      0x0001

/* POSIX-defined flag values accesible to open(2) and/or fcntl(2) */
#define O_NONBLOCK       040000 /* O_NDELAY POSIX style */
#define O_NOCTTY        0100000  /* don't assign controlling tty */
#endif  /* _POSIX_SOURCE */

#ifdef _AUX_SOURCE
/* Implementation-define flag values accessible to open(2) */
#define O_GETCTTY       0200000  /* force controlling tty assignment */
#endif /* _AUX_SOURCE */
#endif /* !__fcntl_h */
```

SEE ALSO

fcntl(2), open(2).

NAME

`font` — description files for device-independent `troff`

SYNOPSIS

`troff -T`*tty-type* ...

DESCRIPTION

For each phototypesetter that `troff(1)` supports and that is available on your system, there is a directory containing files describing the device and its fonts. This directory is named `/usr/lib/font/dev`*tty-type* where *tty-type* is the name of the phototypesetter. Currently, the supported devices are `aps` for the Autologic APS–5, `psc` for a POSTSCRIPT® device such as the Apple LaserWriter®, and `iw` for the Apple ImageWriter® II.

For a particular phototypesetter, *tty-type*, the ASCII file `DESC` in the directory `/usr/lib/font/dev`*tty-type* describes its characteristics. A binary version of the file (described later in this section) is found in the file `/usr/lib/font/dev`*tty-type*`/DESC.out`. Each line of this ASCII file starts with a word that identifies the characteristic which is followed by appropriate specifiers. Blank lines and lines beginning with the # character are ignored.

The legal lines for `DESC` are:

`res` *num*	Resolution of device in basic increments per inch.
`hor` *num*	Smallest unit of horizontal motion.
`vert` *num*	Smallest unit of vertical motion.
`unitwidth` *num*	Point size in which widths are specified.
`sizescale` *num*	Scaling for fractional point sizes.
`paperwidth` *num*	Width of paper in basic increments.
`paperlength` *num*	Length of paper in basic increments.
`biggestfont` *num*	Maximum size of a font.
`sizes` *num num* ...	List of point sizes available on the typesetter.
`fonts` *num name* ...	Number of initial fonts followed by the names of the fonts. For example `fonts 4 R I B S`

`charset` — This always comes last in the file and is on a line by itself. Following it is the list of special character names for this device. Names are separated by a space or a newline. The list can be as long as necessary. Names not in this list are not allowed in the font description files.

`res` is the basic resolution of the device in increments per inch. `hor` and `vert` describe the relationships between motions in the horizontal and vertical directions. If the device is capable of moving in single basic increments in both directions, both `hor` and `vert` would have values of 1. If the vertical motions only take place in multiples of two basic units while the horizontal motions take place in the basic increments, then `hor` would be 1, while `vert` would be 2. `unitwidth` is the point size in which all width tables in the font description files are given. `troff` automatically scales the widths from the `unitwidth` size to the point size it is working with. `sizescale` is not currently used and is 1. `paperwidth` is the width of the paper in basic increments. The APS-5 is 6120 increments wide. `paperlength` is the length of a sheet of paper in the basic increments. `biggestfont` is the maximum number of characters on a font.

For each font supported by the phototypesetter, there is also an ASCII file with the same name as the font (for example, `R`, `I`, `CW`). The format for a font description file is

`name` *name* — Name of the font, such as `R` or `CW`.

`internalname` *name* — Internal name of the font.

`special` — Sets a flag indicating that the font is special.

`ligatures` *name*...`0` — Sets a flag indicating font has ligatures. The list of ligatures follows and is terminated by a zero. Accepted ligatures are: `ff`, `fi`, `fl`, `ffi`, and `ffl`.

`spacewidth` *num* — Specifies width of space if something other than default (1/3 of em) is desired.

`charset` — The `charset` must come at the end. Each line following the word `charset` describes one character in the font. Each line has one of two formats:

name width kerning code
name "

where *name* is either a single ASCII character or a special character name from the list found in DESC. The width is in basic increments. The kerning information is 1 if the character descends below the line, 2 if it rises above the letter ''a,'' and 3 if it both rises and descends. The kerning information for special characters is not used and so may be 0. The code is the number sent to the typesetter to produce the character. The second format is used to indicate that the character has more than one name. The double quote indicates that this name has the same values as the preceding line. The kerning and code fields are not used if the width field is a double quote character. The total number of different characters in this list should not be greater than the value of biggestfont in the DESC file (as described earlier).

troff and its postprocessors read this information from binary files produced from the ASCII files by a program distributed with troff called makedev. For those with a need to know, a description of the format of these files follows.

The file DESC.out starts with the dev structure, defined by dev.h.

```
/*
dev.h: characteristics of a typesetter
*/

struct dev {
short  filesize;      /* number of bytes in file, */
                      /* excluding dev part */
short  res;           /* basic resolution in goobies
                         per inch */
short  hor;           /* goobies horizontally */
short  vert;
```

```
short  unitwidth;    /* size at which widths
                        are given*/
short  nfonts;       /* number fonts physically
                        available */
short  nsizes;       /* number of pointsizes */
short  sizescale;    /* scaling for fractional
                        point sizes */
short  paperwidth;   /* max line length in units */
short  paperlength;  /* max paper length in units */
short  nchtab;       /* number of funny names
                        in chtab */
short  lchname;      /* length of chname table */
short  biggestfont;  /* max # of chars in a font */
short  spare2;       /* in case of expansion */
};
```

`filesize` is merely the size of everything in `DESC.out` excluding the `dev` structure. `nfonts` is the number of different font positions available. `nsizes` is the number of different point sizes supported by this typesetter. `nchtab` is the number of special character names. `lchname` is the total number of characters, including nulls, needed to list all the special character names. At the end of the structure are two spares for later expansion.

Immediately following the `dev` structure are a number of tables. First is the `sizes` table, which contains `nsizes`+1 shorts (a null at the end), describing the point sizes of text available on this device. The second table is the `funny_char_index_table`. It contains indexes for the the table which follows it, the `funny_char_strings`. The indexes point to the beginning of each special character name which is stored in the `funny_char_strings` table. The `funny_char_strings` table is `lchname` characters long, while the `funny_char_index_table` is `nchtab` `shorts` long.

Following the `dev` structure will occur `nfonts` *{font}.out* files, which are used to initialize the font positions. These *{font}.out* files, which also exist as separate files, begin with a `font` structure and then are followed by four character arrays.

```
struct Font {         /* characteristics of a font */
char   nwfont;        /* number of width entries */
char   specfont;      /* 1 == special font */
char   ligfont;       /* 1 == ligatures exist
                         on this font */
char   namefont[10];  /* name of this font,
                         e.g., R */
char   intname[10];   /* internal name of font,
                         in ASCII */
```

`};`

The `font` structure tells how many defined characters there are in the font, whether the font is a "special" font and if it contains ligatures. It also has the ASCII name of the font, which should match the name of the file it appears in, and the internal name of the font located on the typesetting device (*intname*). The internal name is independent of the font position and name that `troff` knows about. For example, you might say "mount `R` in position 4", but when asking the typesetter to actually produce a character from the `R` font, the postprocessor which instructs the typesetter would use *intname*.

The first three character arrays are specific for the font and run in parallel. The first array, `widths`, contains the width of each character relative to `unitwidth`. `unitwidth` is defined in `DESC`. The second array, `kerning`, contains kerning information. If a character rises above the letter "a," 02 is set. If it descends below the line, 01 is set. The third array, `codes`, contains the code that is sent to the typesetter to produce the character.

The fourth array is defined by the device description in `DESC`. It is the `font_index_table`. This table contains indices into the `width`, `kerning`, and `code` tables for each character. The order that characters appear in these three tables is arbitrary and changes from one font to the next. In order for `troff` to be able to translate from ASCII and the special character names to these arbitrary tables, the `font_index_table` is created with an order which is constant for each device. The number of entries in this table is 96 plus the number of special character names for this device. The value 96 is 128-32, the number of printable characters in the ASCII alphabet. To determine whether a normal ASCII character exists, `troff` takes the ASCII value of the character, subtracts 32, and looks in the `font_index_table`. If it finds a 0, the character is not defined in this font. If it finds anything else, that is the index into `widths`, `kerning`, and `codes` tables that describe the character.

To look up a special character name, (for example `\(pl`, the mathematical plus sign), and to determine whether it appears in a particular font or not, the following procedure is followed. A *counter* is set to 0 and an index to a special character name is picked out of the *counter* position in the `funny_char_index_table`. A string comparison is performed between the element in the array funny_char_strings [funny_char_index_table [*counter*]] and the special character name, in our example pl, and if it matches, then troff refers to this character as (96+*counter*). When it wants to determine whether a specific font supports this character, it looks in font_index_table[(96+*counter*)], to see whether there is a 0, meaning the character does not appear in this font, or number, which is the index into the widths, kerning, and codes tables.

Notice that since a value of 0 in the font_index_table indicates that a character does not exist, the 0th element of the width, kerning, and codes arrays are not used. For this reason the 0th element of the width array can be used for a special purpose, defining the width of a space for a font. Normally a space is defined by troff to be 1/3 of the width of the \(em character, but if the 0th element of the width array is nonzero, then that value is used for the width of a space.

SEE ALSO

troff(1).

FILES

/usr/lib/font/dev*tty_type*/DESC.out
/usr/lib/font/dev*tty-type*/*font*.out

NAME
greek — graphics for the extended TTY-37 type-box

SYNOPSIS
cat /usr/pub/greek [| greek -T*terminal*]

DESCRIPTION
greek gives the mapping from ASCII to the "shift-out" graphics in effect between SO and SI on TELETYPE Model 37 terminals equipped with a 128-character type-box. These are the default greek characters produced by nroff. The filters of greek(1) attempt to print them on various other terminals. The file contains:

alpha	α	A	beta	β	B	gamma	γ	\
GAMMA	Γ	G	delta	δ	D	DELTA	Δ	W
epsilon	ε	S	zeta	ζ	Q	eta	η	N
THETA	Θ	T	theta	θ	O	lambda	λ	L
LAMBDA	Λ	E	mu	μ	M	nu	ν	@
xi	ξ	X	pi	π	J	PI	Π	P
rho	ρ	K	sigma	σ	Y	SIGMA	Σ	R
tau	τ	I	phi	φ	U	PHI	Φ	F
psi	ψ	V	PSI	Ψ	H	omega	ω	C
OMEGA	Ω	Z	nabla	∇	[	not	¬	_
partial	∂	]	integral	∫	^			

FILES
/usr/pub/greek

SEE ALSO
300(1), 4014(1), 450(1), greek(1), nroff(1), tc(1).
"Other Text Processing Tools" in *A/UX Text Processing Tools.*

NAME

icmp — Internet Control Message Protocol

SYNOPSIS

None; included automatically with inet(5F).

DESCRIPTION

The Internet Control Message Protocol, ICMP, is used by gateways and destination hosts which process datagrams to communicate errors in datagram-processing to source hosts. The datagram level of Internet is discussed in ip(5P). ICMP uses the basic support of IP as if it were a higher level protocol; however, ICMP is actually an integral part of IP. ICMP messages are sent in several situations; for example: when a datagram cannot reach its destination, when the gateway does not have the buffering capacity to forward a datagram, and when the gateway can direct the host to send traffic on a shorter route.

The Internet protocol is not designed to be absolutely reliable. The purpose of these control messages is to provide feedback about problems in the communication environment, not to make IP reliable. There are still no guarantees that a datagram will be delivered or that a control message will be returned. Some datagrams may still be undelivered without any report of their loss. The higher level protocols which use IP must implement their own reliability mechanisms if reliable communication is required.

The ICMP messages typically report errors in the processing of datagrams; for fragmented datagrams, ICMP messages are sent only about errors in handling fragment 0 of the datagram. To avoid the infinite regress of messages about messages etc., no ICMP messages are sent about ICMP messages. ICMP may however be sent in response to ICMP messages (for example, ECHOREPLY). There are eleven types of ICMP packets which can be received by the system. They are defined in this excerpt from <netinet/ip_icmp.h>, which also defines the values of some additional codes specifying the cause of certain errors. (Comments have been stripped for this listing.)

```
/*
 * Definition of type and code field values
 */
#define ICMP_ECHOREPLY         0
#define ICMP_UNREACH           3
#define  ICMP_UNREACH_NET      0
```

```
#define  ICMP_UNREACH_HOST     1
#define  ICMP_UNREACH_PROTOCOL 2
#define  ICMP_UNREACH_PORT     3
#define  ICMP_UNREACH_NEEDFRAG 4
#define  ICMP_UNREACH_SRCFAIL  5
#define ICMP_SOURCEQUENCH      4
#define ICMP_REDIRECT          5
#define  ICMP_REDIRECT_NET     0
#define  ICMP_REDIRECT_HOST    1
#define  ICMP_REDIRECT_TOSNET  2
#define  ICMP_REDIRECT_TOSHOST 3
#define ICMP_ECHO              8
#define ICMP_TIMXCEED          11
#define  ICMP_TIMXCEED_INTRANS 0
#define  ICMP_TIMXCEED_REASS   1
#define ICMP_PARAMPROB         12
#define ICMP_TSTAMP            13
#define ICMP_TSTAMPREPLY       14
#define ICMP_IREQ              15
#define ICMP_IREQREPLY         16
```

Arriving ECHO and TSTAMP packets cause the system to generate ECHOREPLY and TSTAMPREPLY packets. IREQ packets are not yet processed by the system, and are discarded. UNREACH, SOURCEQUENCH, TIMXCEED and PARAMPROB packets are processed internally by the protocols implemented in the system, or reflected to the user if a raw socket is being used; see ip(5P). REDIRECT, ECHOREPLY, TSTAMPREPLY and IREQREPLY are also reflected to users of raw sockets. In addition, REDIRECT messages cause the kernel routing tables to be updated; see routing(5N).

SEE ALSO

inet(5F), ip(5P).
Internet Control Message Protocol, RFC792, J. Postel, USC-ISI

BUGS

IREQ messages are not processed properly: the address fields are not set.

Messages which are source routed are not sent back using inverted source routes, but rather go back through the normal routing mechanisms.

NAME

inet — Internet protocol family

SYNOPSIS

```
#include <sys/types.h>
#include <netinet/in.h>
```

DESCRIPTION

The Internet protocol family is a collection of protocols layered atop the Internet Protocol (IP) transport layer, and utilizing 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 (on the VAX these are word and byte reversed). The include file <netinet/in.h> defines this address as a discriminated union.

Sockets bound to the Internet protocol family utilize 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 address INADDR_ANY to effect ‘‘wildcard’’ matching on incoming messages.

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 while 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 not directly accessible.

SEE ALSO

tcp(5P), udp(5P), ip(5P).

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>
```

DESCRIPTION

IP is the transport layer protocol used by the Internet protocol family. It may be accessed through a "raw socket" when developing new protocols, or special purpose applications. IP sockets are connectionless, and are normally used with the sendto and recvfrom calls, though the connect(2N) call may also be used to fix the destination for future packets (in which case the read(2) or recv(2N) and write(2) or send(2N) system calls may be used).

Outgoing packets automatically have an IP header prefixed to them (based on the destination address and the protocol number the socket is created with). Likewise, incoming packets have their IP header stripped before being sent to the user.

ERRORS

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.
[ENOBUFS]	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.

SEE ALSO

send(2N), recv(2N), intro(5), inet(5F).

BUGS

One should be able to send and receive `ip` options.

The protocol should be settable after socket creation.

NAME

`lo` — software loopback network interface

SYNOPSIS

pseudo-device `loop`

DESCRIPTION

The `loop` interface is a software loopback mechanism which may be used for performance analysis, software testing, and/or local communication. By default, the loopback interface is accessible at address 127.0.0.1 (nonstandard); this address may be changed with the `SIOCSIFADDR` `ioctl`.

DIAGNOSTICS

`lo%d: can't handle af%d`. The interface was handed a message with addresses formatted in an unsuitable address family; the packet was dropped.

SEE ALSO

`intro`(5), `inet`(5F).

BUGS

It should handle all address and protocol families. An approved network address should be reserved for this interface.

NAME

man — macros for formatting entries in this manual

SYNOPSIS

`nroff -man` *files*

`troff -man [-rs1]` *files*

DESCRIPTION

These `nroff`(1)/ `troff`(1) macros are used to lay out the format of the entries of this manual. The default page size is 8.5″×11″, with a 6.5″×10″ text area; the `-rs1` flag option reduces these dimensions to 6″×9″ and 4.75″×8.375″, respectively; this option (which is *not* effective in `nroff`(1)) also reduces the default type size from 10-point to 9-point, and the vertical line spacing from 12-point to 10-point. The `-rV2` flag option may be used to set certain parameters to values appropriate for certain Versatec printers: it sets the line length to 82 characters, the page length to 84 lines, and it inhibits underlining.

Any *text* argument below may be one to six "words". Double quotes ("") may 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` may be used to italicize a whole line, or `.SM` followed by `.B` to make small bold text. By default, hyphenation is turned off for `nroff`(1), but remains on for `troff`(1).

Type font and size are reset to default values before each paragraph and after processing font- and size-setting macros, e.g., `.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`(1), 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, e.g., "local," *n* is new manual name. Invokes `.DT` (see below).
`.SH` *text*	Place subhead *text*, e.g., SYNOPSIS, here.
`.SS` *text*	Place sub-subhead *text*, e.g., "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; see mm(1).
.DT	Restore default tab settings (every 7.2 ens in troff(1), 5 ens in nroff(1)).
.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(1), 1v in nroff(1)).

The following *strings* are defined:

*R	® in troff(1), (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(1), 5 ens in nroff(1)).
LL	Line length including IN.
PD	Current interparagraph distance.

EXAMPLES

The `man` macros are provided to process manual pages already on-line at a given location and to enable users to make their own manual pages. The preceding section demonstrated the usage of the macros themselves; the following section provides examples of command lines typically used to process the completed files.

`man` macros are designed to run with either `nroff` or `troff`. The first command line will process a file using only macros and `nroff` requests:

```
nroff -Tlp -man file | lp
```

The file is piped to the local line printer, `lp`.

The next command line will process a file containing tables as well as macros and `nroff` requests:

```
tbl | nroff -Tlp -man file | col | lp
```

Notice that before it is sent to the line printer, the output is first filtered through `col`, to process the reverse line feeds used by `tbl`.

The final example is a command line that processes an unusual manual page, one using `pic`. If the manual pages created with `man` are intended for an on-line facility, components requiring `troff`, such as `pic` (or `grap`) should be avoided since the average installation of terminals will not be able to process typeset documents.

`pic file | tbl | troff -Taps -man |` *typesetter*

`grap` precedes `pic` because it is a preprocessor to `pic`; the reverse order, of course, will not format correctly. The file contains one or more tables, requiring `tbl`, but `col` is no longer necessary because typeset documents do not use reverse line feeds with which to make tables. The `-T` flag option for specifying the output device (terminal type) takes the argument `aps` here, readying the document for processing on the APS-5 phototypesetter.

CAVEATS

Special macros, strings, and number registers exist, internal to `man`, in addition to those mentioned above. Except for names predefined by `troff`(1) 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.

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 (`CW`). Of course, if the input text of an entry contains requests for other fonts (e.g., `.I`, `.RB`, `\fI`), the corresponding fonts must be mounted.

FILES

`/usr/lib/tmac/tmac.an`
`/usr/lib/macros/cmp.n.[dt].an`
`/usr/lib/macros/ucmp.n.an`

SEE ALSO

`eqn`(1), `man`(1), `tbl`(1), `tc`(1), `troff`(1).
"Other Text Processing Tools" in *A/UX Text Processing Tools*.

BUGS

If the argument to `.TH` contains *any* blanks and is *not* enclosed by double quotes (`""`), there will be strange irregular dots on the output.

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 double-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	The ratio of the circumference of a circle to its diameter. (There are also several fractions of its reciprocal and its square root.)
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 [*nroff-options...*]
troff –me [*troff-options...*]

DESCRIPTION

me is a package of nroff and troff macro definitions that provides a canned 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; however, these requests may be used with impunity after the first .pp:

.bp	Begin a new page.
.br	Break the 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 the next *n* lines.
.ul *n*	Underline the next *n* lines.
.sz +*n*	Add *n* to the point size.

Output of the eqn, neqn, refer, and tbl 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).

A/UX Text Processing Tools.

REQUESTS

In the following list, initialization refers to the first .pp, .lp, .ip, .np, .sh, or .uh macro. This list is incomplete.

MACRO NAME	INITIAL VALUE	BREAK? RESET?	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.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, anabstract, table of contents, and so on.), B (bibliography), RC (chapters renumbered from page of one each chapter), or RA (appendix renumbered from page one).
.+c *T*	-	yes	Begin chapter (appendix, and so on, as set by .++). . *T* is the chapter title.
.1c	1	yes	One-column format on a new page.
.2c	1	yes	Two-column format.
.EN	-	yes	Space after equation produced by eqn or neqn.
.EQ *x y*	-	yes	Precede equation; break out and add space. The equation number is y. The optional argument *x* may be *I* to indent the 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* the table has a 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	Augment 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 (no-fill 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*. Indentation is *y* ens (default 5).
.lp	yes	yes	Start left-blocked paragraph.
.lo	-	no	Read in a file of local macros of the form * *x*. Must be given before initialization.
.np	1	yes	Start numbered paragraph.
.of'x'y'z	''''	no	Set odd footer to x y z.
.oh'x'y'z	''''	no	Set odd header to x y z.
.pd	-	yes	Print delayed text.
.pp	no	yes	Begin paragraph with the 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 x	-	yes	Section head follows, font automatically bold. *n* is the level of section, and *x* is the title of the section.
.sk	no	no	Leave the next page blank. Only one page is remembered ahead.
.sm *x*	-	no	Set *x* in a smaller point size.
.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. (No-fill only).
.uh	-	yes	Like .sh but unnumbered.
.xp *x*	-	no	Print index *x*.

NAME

mm — macro package for formatting documents

SYNOPSIS

mm [*options*] [*files*]

nroff -mm [*options*] [*files*]

nroff -cm [*options*] [*files*]

mmt [*options*] [*files*]

troff -mm [*options*] [*files*]

DESCRIPTION

This package provides a formatting capability for a very wide variety of documents. The manner in which you type and edit a document is essentially independent of whether the document is to be eventually formatted at a terminal or is to be phototypeset.

Full details are provided in *A/UX Text Processing Tools.*

FILES

/usr/lib/tmac/tmac.m	pointer to the noncompacted version of the package
/usr/lib/macros/mm[nt]	noncompacted version of the package

SEE ALSO

mm(1), mmt(1), nroff(1), troff(1).
"mm Reference" in *A/UX Text Processing Tools.*

NAME

mptx — the macro package for formatting a permuted index

SYNOPSIS

`nroff -mptx` [*options*] [*files*]

`troff -mptx` [*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` flag option must be invoked *after* the `-mm` call. For example:

`nroff -mm -mptx` *file*

or

`mm -mptx` *file*

FILES

`/usr/lib/tmac/tmac.ptx`	pointer to the macro package
`/usr/lib/macros/ptx`	macro package

SEE ALSO

`mm`(1), `nroff`(1), `ptx`(1), `troff`(1), `mm`(5).
"Other Text Processing Tools" in *A/UX Text Processing Tools*.

NAME

ms — text formatting macros

SYNOPSIS

`nroff -ms` [*nroff-options*...]

`troff -ms` [*troff-options*...]

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`(1). All external `ms` macros are defined below. Many `nroff` and `troff` requests are unsafe in conjunction with this package. However, the first four requests below may be used with impunity after initialization, and the last two may be used even before initialization:

`.bp`	begin new page
`.br`	break output line
`.sp` *n*	insert *n* spacing lines
`.ce` *n*	center next *n* lines
`.ls` *n*	line spacing: *n*=1 single, *n*=2 double space
`.na`	no alignment of right margin

Font and point size changes with `\f` and `\s` are also allowed; for example, `\fIword\fP` will produce *word*. Output of the `tbl`, `eqn`, and `refer`(1) preprocessors for equations, tables, and references is acceptable as input.

Full details are provided in *A/UX Text Processing Tools.*

FILES

`/usr/lib/tmac/tmac.x`
`/usr/lib/ms/x.???`

SEE ALSO

`eqn`(1), `refer`(1), `tbl`(1), `troff`(1).
"`ms` Reference" in *A/UX Text Processing Tools.*

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

.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	thesis mode
.TS *x*	–	y,y	begin table; if *x*=H table has multi-page header
.UL *x*	–	n	underline *x* (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
.]-	–	n	beginning of refer reference
.[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:

PS	point size	paragraph	10
VS	vertical spacing	paragraph	12
LL	line length	paragraph	6i
LT	title length	next page	same as LL
FL	footnote length	next .FS	5.5i
PD	paragraph distance	paragraph	1v (if n), .3v (if t)
DD	display distance	displays	1v (if n), .5v (if t)
PI	paragraph indent	paragraph	5n
QI	quote indent	next .QP	5n
FI	footnote indent	next .FS	2n

PO	page offset	next page	0 (if n), ~1i (if t)
HM	header margin	next page	1i
FM	footer margin	next page	1i
FF	footnote format	next .FS	0 (1, 2, 3 available)

When resetting these values, make sure to specify the appropriate units. Setting the line length to 7, for example, will result in output with one character per line. Setting FF to 1 suppresses footnote superscripting; setting it to 2 also suppresses indentation of the first line; and setting it to 3 produces an .IP-like footnote paragraph.

Here is a list of string registers available in ms; they may be used anywhere in the text:

NAME	STRING'S FUNCTION
*Q	quote (" in nroff, “ in troff)
*U	unquote (" in nroff, ” in troff)
*-	dash (-- in nroff, — in troff)
*(MO	month (month of the year)
*(DY	day (current date)
**	automatically numbered footnote
*´	acute accent (before letter)
*`	grave accent (before letter)
*^	circumflex (before letter)
*,	cedilla (before letter)
*:	umlaut (before letter)
*~	tilde (before letter)

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 viewgraphs and slides

SYNOPSIS

mvt [-a] [*options*] [*files*]

troff [-a] [-rX1] -mv [*options*] [*files*]

DESCRIPTION

This package makes it easy to typeset viewgraphs 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), eqn(1), tbl(1), pic(1), and grap(1) are available for more difficult tasks.

The output can be previewed on most terminals, and, in particular, on the TEKTRONIX 4014. For this device, specify the -rX1 option (this option is automatically specified by the mvt command when that command is invoked with the -D4014 option). 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 viewgraphs 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 viewgraphs: the ratio of the longer dimension to the shorter one is larger for slides than for viewgraphs. As a result, slide foils can be used for viewgraphs, but not vice versa; on the other hand, viewgraphs 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 oversize, 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 nonnull 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 uppercase synonyms for the corresponding lowercase `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

`eqn`(1), `mmt`(1), `tbl`(1), `troff`(1).
"Other Text Processing Tools" in *A/UX Text Processing Tools.*

NAME

nterm — terminal driving tables for nroff

DESCRIPTION

nroff(1) uses driving tables to customize its output for various types of output devices, such as printing terminals, special word processing terminals (such as Diablo, Qume, or NEC Spinwriter mechanisms), or special output filter programs. These driving tables are written as ASCII files, and are installed in /usr/lib/nterm/tab.name, where name is the name for that terminal type as given in term(5).

The first line of a driving table should contain the name of the terminal: simply a string with no embedded white space. ''white space'' means any combination of spaces, tabs and newlines. The next part of the driver table is structured as follows:

bset [*integer*] (not supported in all versions of nroff)
breset [*integer*] (not supported in all versions of nroff)
Hor [*integer*]
Vert [*integer*]
Newline [*integer*]
Char [*integer*]
Em [*integer*]
Halfline [*integer*]
Adj [*integer*]
twinit [*character-string*]
twrest [*character-string*]
twnl [*character-string*]
hlr [*character-string*]
hlf [*character-string*]
flr [*character-string*]
bdon [*character-string*]
bdoff [*character-string*]
iton [*character-string*]
itoff [*character-string*]
ploton [*character-string*]
plotoff [*character-string*]
up [*character-string*]
down [*character-string*]
right [*character-string*]
left [*character-string*]

The meanings of these fields are as follows:

`bset`	bits to set in the `c_oflag` field of the `termio` structure before output.
`breset`	bits to reset in the `c_oflag` field of the `termio` structure before output.
`Hor`	horizontal resolution in units of 1/240 of an inch.
`Vert`	vertical resolution in units of 1/240 of an inch.
`Newline`	space moved by a newline (linefeed) character in units of 1/240 of an inch.
`Char`	quantum of character sizes, in units of 1/240 of an inch. (That is, a character is a multiple of `Char` units wide)
`Em`	size of an em in units of 1/240 of an inch.
`Halfline`	space moved by a half-linefeed (or half-reverse-linefeed) character in units in 1/240 of an inch.
`Adj`	quantum of white space, in 1/240 of an inch. (i.e., white spaces are a multiple of Adj units wide) *Note:* If this is less than the size of the space character, `nroff` will output fractional spaces using plot mode. Also, if the `-e` switch to `nroff` is used, `Adj` is set equal to `Hor` by `nroff`.
`twinit`	sequence of characters used to initialize the terminal in a mode suitable for `nroff`.
`twrest`	sequence of characters used to restore the terminal to normal mode.
`twnl`	sequence of characters used to move down one line.
`hlr`	sequence of characters used to move up one-half line.
`hlf`	sequence of characters used to move down one-half line.

`flr`	sequence of characters used to move up one line.
`bdon`	sequence of characters used to turn on hardware boldface mode, if any.
`bdoff`	sequence of characters used to turn off hardware boldface mode, if any.
`iton`	sequence of characters used to turn on hardware italics mode, if any.
`itoff`	sequence of characters used to turn off hardware italics mode, if any.
`ploton`	sequence of characters used to turn on hardware plot mode (for Diablo type mechanisms), if any.
`plotoff`	sequence of characters used to turn off hardware plot mode (for Diablo type mechanisms), if any.
`up`	sequence of characters used to move up one resolution unit (`Vert`) in plot mode, if any.
`down`	sequence of characters used to move down one resolution unit (`Vert`) in plot mode, if any.
`right`	sequence of characters used to move right one resolution unit (`Hor`) in plot mode, if any.
`left`	sequence of characters used to move left one resolution unit (`Hor`) in plot mode, if any.

This part of the driving table is fixed format, and you cannot change the order of entries. You should put entries on separate lines, and these lines should contain exactly two fields (no comments allowed) separated by white space. For example,

```
Cbset   0
breset  0
Hor     24
```

and so on.

Follow this first part of the driving table with a line containing the word "`charset`," and then specify a table of special characters that you want to include. That is, specify all the non-ASCII characters that `nroff`(1) knows by two character names, such as -. If `nroff` does not find the word ``charset'' where it expects to, it will abort with an error message.

Each definition in the part after ``charset'' occupies one line, and has the following format:

chname width output

where ``*chname*'' is the (two letter) name of the special character, ``*width*'' is its width in ems, and ``*output*'' is the string of characters and escape sequences to send to the terminal to produce the special character.

If any field in the ``charset'' part of the driving table does not pertain to the output device, you may give that particular sequence as a null string, or leave out the entry. Special characters that do not have a definition in this file are ignored on output by nroff(1).

You may put the ``charset'' definitions in any order, so it is possible to speed up nroff by putting the most used characters first. For example,

```
charset
em 1 -
hy 1 -
\- 1 -
bu 1 +
```

and so on.

The best way to create a terminal table for a new device is to take an existing terminal table and edit it to suit your needs. Once you create such a file, put it in the directory /usr/lib/nterm, and give it the name tab.*xyz* where *xyz* is the name of the terminal and the name that you pass nroff via the -T flag option (for example, nroff -T*xyz*).

FILES

/usr/lib/nterm/tab.name

SEE ALSO

nroff(1).

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 up to six letters, numbers or underscores. Each *name* in a single compilation must be unique, but may be the same as any ordinary program symbol.

For marks to be effective, the symbol MARK must be defined before the header file `<prof.h>` is included. 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.

EXAMPLES

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) errors-code

#include <regexp.h>

char *compile(instring, expbuf, endbuf, eof)
char *instring, *expbuf, *endbuf;
int  eof ;

int step(string, exbuf)
char *string, *exbuf;

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 /usr/include/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 that 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 which 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 which call functions to input characters or have characters in an external array can pass down a value of `((char *) 0)` for this parameter.

The next parameter *expbuf* is a character pointer. It points to the place where the compiled regular expression will be placed.

The parameter *endbuf* is one more than the highest address where the compiled regular expression may be placed. If the compiled expression cannot fit in (*endbuf–expbuf*) bytes, a call to ERROR(50) is made.

The parameter *eof* is the character which marks the end of the regular expression. 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 sometime 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

bs(1), ed(1), expr(1), grep(1), sed(1).

BUGS

The handling of circf is awkward.

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;
        ino_t   st_ino;
        ushort  st_mode;
        short   st_nlink;
        short   st_uid;
        short   st_gid;
        dev_t   st_rdev;
        off_t   st_size;
        time_t  st_atime;
        int     st_spare1;
        time_t  st_mtime;
        int     st_spare2;
        time_t  st_ctime;
        int     st_spare3;
        long    st_blksize;
        long    st_blocks;
        long    st_spare4[2];
};

#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_IFLNK 0120000     /* symbolic link */
#define S_IFSOC 0140000     /* socket */
#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_IWRIT 00200       /* write permission, owner */
```

```
#define S_IEXEC 00100      /* execute/search permission,
                              owner */
#define S_IRUSR 00400      /* read permission, owner */
#define S_IWUSR 00200      /* write permission, owner */
#define S_IXUSR 00100      /* execute/search permission,
                              owner */
#define S_IRWXU (S_IRUSR | S_IWUSR | S_IXUSR)

#define S_IRGRP 00040      /* read permission, group */
#define S_IWGRP 00020      /* write permission, group */
#define S_IXGRP 00010      /* execute/search permission,
                              group */
#define S_IRWXG (S_IRGRP | S_IWGRP | S_IXGRP)

#define S_IROTH 00004      /* read permission, others */
#define S_IWOTH 00002      /* write permission, others */
#define S_IXOTH 00001      /* execute/search permission,
                              others */
#define S_IRWXO (S_IROTH | S_IWOTH | S_IXOTH)
```

FILES

/usr/include/sys/types.h
/usr/include/sys/stat.h

SEE ALSO

stat(2), types(5).

NAME

`tcp` — Internet Transmission Control Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/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 utilizing 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`(2N) system call must be used after binding the socket with the `bind`(2N) system call. Only passive sockets may use the `accept`(2N) call to accept incoming connections. Only active sockets may use the `connect`(2N) call to initiate connections.

Passive sockets may "underspecify" their location to match incoming connection requests from multiple networks. This technique, termed "wildcard addressing," allows a single server to provide service to clients on multiple networks. To create a socket which listens on all networks, the Internet address `INADDR_ANY` must be bound. The TCP port may 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.

ERRORS

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;

[ENOBUFS]	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.

SEE ALSO

intro(5), inet(5F).

BUGS

It should be possible to send and receive TCP options. The system always tries to negotiates the maximum TCP segment size to be 1024 bytes. This can result in poor performance if an intervening network performs excessive fragmentation.

NAME

term — conventional names for terminals

DESCRIPTION

These names are used by certain commands (e.g., nroff(1), mm(1), man(1), tabs(1)) and are maintained as part of the shell environment (see sh(1), profile(4), and environ(5)) in the variable $TERM:

1520	Datamedia 1520
1620	Diablo 1620 and others using the HyType II printer
1620–12	same, in 12-pitch mode
2621	Hewlett-Packard HP2621 series
2631	Hewlett-Packard 2631 line printer
2631–c	Hewlett-Packard 2631 line printer - compressed mode
2631–e	Hewlett-Packard 2631 line printer - expanded mode
2640	Hewlett-Packard HP2640 series
2645	Hewlett-Packard HP264n series (other than the 2640 series)
300	DASI/DTC/GSI 300 and others using the HyType I printer
300–12	same, in 12-pitch mode
300s	DASI/DTC/GSI 300s
382	DTC 382
300s–12	same, in 12-pitch mode
3045	Datamedia 3045
33	TELETYPE Terminal Model 33 KSR
37	TELETYPE Terminal Model 37 KSR
40–2	TELETYPE Terminal Model 40/2
40–4	TELETYPE Terminal Model 40/4
4540	TELETYPE Terminal Model 4540
3270	IBM Model 3270
4000a	Trendata 4000a
4014	Tektronix 4014
43	TELETYPE Model 43 KSR
450	DASI 450 (same as Diablo 1620)
450–12	same, in 12-pitch mode
735	Texas Instruments TI735 and TI725
745	Texas Instruments TI745
dumb	generic name for terminals that lack reverse linefeed and other special escape sequences
sync	generic name for synchronous TELETYPE 4540-compatible terminals
hp	Hewlett-Packard (same as 2645)
lp	generic name for a line printer

tn1200 General Electric TermiNet 1200
tn300 General Electric TermiNet 300

Up to 8 characters, chosen from [–a–z0–9], make up a basic terminal name. Terminal submodels and operational modes are distinguished by suffixes beginning with a –. 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.

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`.

See `/etc/termcap` on your system for a complete list.

SEE ALSO

`mm`(1), `nroff`(1), `sh`(1), `stty`(1), `tabs`(1), `tplot`(1G), `profile`(4), `environ`(5).

BUGS

This is a small candle trying to illuminate a large, dark problem. Programs that ought to adhere to this nomenclature do so somewhat fitfully.

NAME

troff — description of troff output language

DESCRIPTION

The device-independent troff outputs a pure ASCII description of a typeset document. The description specifies the typesetting device, the fonts, and the point sizes of characters to be used as well as the position of each character on the page. A list of all the legal commands follows. Most numbers are denoted as *n* and are ASCII strings. Strings inside of brackets ([]) are optional. troff may produce them, but they are not required for the specification of the language. The character \n has the standard meaning of ''newline'' character. Between commands, white space has no meaning. White space characters are spaces and newlines.

s*n* The point size of the characters to be generated.

f*n* The font mounted in the specified position is to be used. The number ranges from 0 to the highest font presently mounted. 0 is a special position, invoked by troff, but not directly accessible to the troff user. Normally fonts are mounted starting at position 1.

c*x* Generate the character *x* at the current location on the page; *x* is a single ASCII character.

C*xyz* Generate the special character *xyz*. The name of the character is delimited by white space. The name will be one of the special characters legal for the typesetting device as specified by the device specification found in the file DESC. This file resides in a directory specific for the typesetting device. (See font(5) and /usr/lib/font/dev*.)

H*n* Change the horizontal position on the page to the number specified. The number is in basic units of motions as specified by *DESC*. This is an absolute ''goto''.

h*n*	Add the number specified to the current horizontal position. This is a relative "goto".
V*n*	Change the vertical position on the page to the number specified (down is positive).
v*n*	Add the number specified to the current vertical position.
nnx	This is a two-digit number followed by an ASCII character. The meaning is a combination of h*n* followed by c*x*. The two digits *nn* are added to the current horizontal position and then the ASCII character, *x*, is produced. This is the most common form of character specification.
n*b a*	This command indicates that the end of a line has been reached. No action is required, though by convention the horizontal position is set to 0. troff will specify a resetting of the *x*,*y* coordinates on the page before requesting that more characters be printed. The first number, *b*, is the amount of space before the line and the second number, *a*, the amount of space after the line. The second number is delimited by white space.
w	A w appears between words of the input document. No action is required. It is included so that one device can be emulated more easily on another device.
p*n*	Begin a new page. The new page number is included in this command. The vertical position on the page should be set to 0.
# .... \n	A line beginning with a pound sign is a comment.
Dl *x* y\n	Draw a line from the current location to *x*,*y*.

`Dc` *d*`\n`	Draw a circle of diameter *d* with the left-most edge being at the current location (x, y). The current location after drawing the circle will be x+*d*,y, the rightmost edge of the circle.
`De` *dx dy*`\n`	Draw an ellipse with the specified axes. *dx* is the axis in the x direction and *dy* is the axis in the y direction. The leftmost edge of the ellipse will be at the current location. After drawing the ellipse the current location will be x+*dx*,y.
`Da` *x y u v*	Draw a counterclockwise arc from the current location to *x+u,y+v* using a circle of whose center is *x,y* from the current location. The current location after drawing the arc will be at its end.
`D~` *x y x y...*`\n`	Draw a spline curve (wiggly line) between each of the *x,y* coordinate pairs starting at the current location. The final location will be the final *x,y* pair of the list.
`x i[nit]\n`	Initialize the typesetting device. The actions required are dependent on the device. An `init` command will always occur before any output generation is attempted.
`x T` *device*`\n`	The name of the typesetter is *device*. This is the same as the argument to the `-T` option. The information about the typesetter will be found in the directory `/usr/lib/font/dev`*{device}*.
`x r[es]` *n h v*`\n`	The resolution of the typesetting device in increments per inch is *n*. Motion in the horizontal direction can take place in units of *h* basic increments. Motion in the vertical direction can take place in units of *v* basic increments. For example, the APS-5 typesetter has a basic resolution of 723 increments per inch and can move in either direction in 723rds of an inch. Its

	specification is: `x res 723 1 1`
`x p[ause]\n`	Pause. Cause the current page to finish but do not relinquish the typesetter.
`x s[top]\n`	Stop. Cause the current page to finish and then relinquish the typesetter. Perform any shutdown and bookkeeping procedures required.
`x t[railer]\n`	Generate a trailer. On some devices no operation is performed.
`x f[ont]` *n name*`\n`	Load the font *name* into position *n*.
`x H[eight]` *n*`\n`	Set the character height to *n* points. This causes the letters to be elongated or shortened. It does not affect the width of a letter.
`x S[lant]` *n*`\n`	Set the slant to *n* degrees. Only some typesetters can do this and not all angles are supported.

SEE ALSO

troff(1).
"nroff/troff Reference" and "Introduction to troff and mm" in *A/UX Text Processing Tools.*

NAME

types — primitive system data types

SYNOPSIS

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

DESCRIPTION

The data types defined in the include file are used in A/UX® system code; some data of these types are accessible to user code:

```
#ifndef __sys_types_h
#define __sys_types_h
/*
  * System-dependent parameters and types
*/

typedef    char *caddr_t;
typedef    long clock_t;
typedef    short cnt_t;
typedef    long daddr_t;
typedef    unsigned short dev_t;
typedef    short gid_t;
typedef    unsigned short ino_t;
typedef    long key_t;
typedef    int label_t[13];
typedef    unsigned short mode_t;
typedef    short nlink_t;
typedef    long off_t;
typedef    long paddr_t;
typedef    int pid_t;
typedef    int ptrdiff_t;
typedef    int size_t;
typedef    long time_t;
typedef    long ubadr_t;
typedef    unsigned char uchar_t;
typedef    unsigned short ushort_t;
typedef    short uid_t;
typedef    unsigned int uint_t;
typedef    unsigned long ulong_t;
typedef    unsigned int wchar_t;

#ifndef NULL
#define NULL      0
#endif  /* NULL */
```

```
/*
  *To be excluded from visibility control,
   types must end in _t.
 */
#ifdef _SYSV_SOURCE
typdef     unsigned int  uint;
typdef     unsigned long       ulong;
typdef     unsigned char       unchar;
typdef     unsigned short      ushort;
#endif /* _SYSV_SOURCE */

#ifdef _BSD_SOURCE
typedef     struct fd_set { long fds_bits[1]; } fd_set;
typedef     struct{int r[1];} *physadr;
typedef     struct _quad{ long val[2]; } quad;
typedef     unsigned char      u_char;
typedef     unsigned short     u_short;
typedef     unsigned int       u_int;
typedef     unsigned long      u_long;
#endif /* _BSD_SOURCE */

#ifdef _AUX_SOURCE
typedef     unisigned long ino_tl;
#endif /* _AUX_SOURCE */
#endif /* !__sys_types_h */
```

The form `daddr_t` is used for disk addresses except in an inode 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 and are installation-dependent. Offsets are measured in bytes from the beginning of a file. The `label_t` variables are used to save the processor state while another process is running.

SEE ALSO

`fs`(4).

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 which 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, though the connect(2N) call may also be used to fix the destination for future packets (in which case the recv(2N) or send(2N) system calls may be used).

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 (i.e., a UDP port may not be "connected" to a TCP port). In addition broadcast packets may be sent (assuming the underlying network supports this) by using a reserved "broadcast address"; this address is network interface dependent.

ERRORS

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;
[ENOBUFS]	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 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.

SEE ALSO

send(2N), recv(2N), intro(5), inet(5F).

NAME

`values` — machine-dependent values

SYNOPSIS

```
#include <values.h>
```

DESCRIPTION

This file contains a set of manifest constants, conditionally defined for particular processor architectures.

The model assumed for integers is binary representation (one's or two's complement), where the sign is represented by the value of the high-order bit.

`BITS` (*type*)
: The number of bits in a specified type (for example, `int`).

`HIBITS`
: The value of a short integer with only the high-order bit set (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).

1027045417

THE APPLE PUBLISHING SYSTEM

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh® computers and troff running on A/UX. Proof and final pages were created on Apple LaserWriter® printers. POSTSCRIPT®, the page-description language for the LaserWriter, was developed by Adobe Systems Incorporated.

Text type and display type are Times and Helvetica. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Apple Courier.

Writers: J. Eric Akin, Mike Elola, George Towner, and Kathy Wallace

Editor: George Truett

Production Supervisor: Josephine Manuele

Acknowledgments: Lori Falls and Michael Hinkson

Special thanks to Lorraine Aochi, Vicki Brown, Sharon Everson, Pete Ferrante, Kristi Fredrickson, Don Gentner, Tim Monroe, Dave Payne, Henry Seltzer, and John Sovereign

030-0785