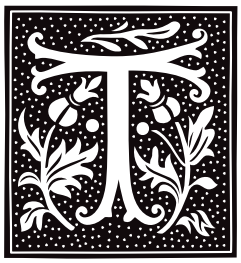

tail — Command

Print the end of a file
tail [+*n***[bcfl]] [***file***]**
tail [-*n***[bcfl]] [***file***]**

tail copies the last part of *file*, or of the standard input if none is named, to the standard output.

The given *number* tells **tail** where to begin to copy the data. Numbers of the form *+number* measure the starting point from the beginning of the file; those of the form *-number* measure from the end of the file.

A specifier of blocks, characters, or lines (**b**, **c**, or **l**, respectively) may follow the number; the default is lines. If no *number* is specified, a default of -10 is assumed.

The **-f** option opens the tail of a file, and then displays new material as it is added to a file. This command lets you watch a file as it is being built, such as by **nroff**. Note that when **tail** is invoked with this option, it does not exit; therefore, when you wish to exit, type the interrupt character (usually **<ctrl-C>**).

See Also

commands, **dd, egrep, head, sed**

Notes

Because **tail** buffers data measured from the end of the file, large counts may not work.

tan() — Mathematics Function (libm)

Calculate tangent
#include <math.h>
double tan(*radian***) double** *radian***;**

tan() calculates the tangent of its argument *radian*, which must be in radian measure.

Example

The following program implements the Fresnel equation, which computes the percentage of light or energy reflected from perfect glass, based on the angle of incidence. It is by Dmitry Gringauz (dmitry@golem.com). Be sure to compile it with the options **-f** and **-lm**.

```
#include <math.h>
#include <stdio.h>

double deg_to_rad(deg)
double deg;
{
      return deg*PI/180.0;
}

double rad_to_deg(rad)
double rad;
{
      return rad*180.0/PI;
}
```

```
main()
{
      double i=0.0; /* incidence angle */
      double Ra=0.0; /* angle of refraction */
      double Rho=0.0; /* % reflection of the beam */
      double Ri=1.52; /* refractive index of glass */

      printf("\tAngle\t\tRho\n");
      printf("\t-----\t\t---\n");

      for (i = 5.0; i <= 90.0; i = i+5.0) {
            double x = 0.0, y = 0.0; /* temporaries */

            /* find the angle of refraction */
            Ra = rad_to_deg(asin( sin(deg_to_rad(i)) / Ri));

            /* makes sense to calculate these only once */
            x = deg_to_rad(i - Ra);
            y = deg_to_rad(i + Ra);

            /* find out percent of reflected energy */
            Rho = pow(sin(x), 2.0) / pow(sin(y), 2.0) +
                        pow(tan(x), 2.0) / pow(tan(y), 2.0);
            Rho = Rho/2.0*100.00;
            printf("\t%f\t%f\n", i, Rho);
      } /* for */
} /* main */
```

See Also

libm, tanh()
ANSI Standard, §7.5.2.7
POSIX Standard, §8.1

Diagnostics

tan() returns a very large number where it is singular, and sets **errno** to **ERANGE**.

tanh() — Mathematics Function (libm)

Calculate hyperbolic cosine
#include <math.h>
double tanh(*radian***) double** *radian***;**

tanh() calculates the hyperbolic tangent of *radian*, which is in radian measure.

See Also

libm, tan()
ANSI Standard, §7.5.3.3
POSIX Standard, §8.1

Diagnostics

When an overflow occurs, **tanh()** sets **errno** to **ERANGE**.

tape — Technical Information

Magnetic-tape devices

The COHERENT system supports two classes of magnetic-tape devices: *floppy tape*, in which the tape device is plugged into your system's floppy-disk controller; and *SCSI tape*, in which the tape device is plugged into your system's SCSI controller (should it have one). The following gives general remarks on tape devices, then briefly discusses the drivers for tape devices and the block-special files by which you can access them.

Tape Devices

A tape consists of one or more files. Each file, in turn, consists of one or more records and is terminated by a tape mark. Two tape marks terminate the last file. Tape records may vary in length, but cannot exceed 32 kilobytes (16 kilobytes is more practical).

Like other block-oriented devices, tape units can be accessed through a system's *cooked* interface or its *raw* interface. On a cooked device, seeking to any byte offset and reading in any number of bytes is possible. You

cannot read beyond the tape mark at the end of the current file. For block-I/O requests, every record in the file must be 512 bytes long. Write requests must be made in increments of 512 bytes.

A raw device bypasses the buffer cache, so that data are written directly to or from your buffer. One write request generates one tape record, and one read request returns exactly one record. The number of bytes read may be fewer than expected. If the tape mark is read, a count of zero is returned, but the system positions the tape at the start of the next tape file. Seeking on a raw device is ignored, and mounting is not allowed. Raw (or character) requests are usually performed in units much larger than 512 bytes.

A unit cannot be opened if it is off-line or already in use. If tape cartridge within the tape drive is write protected, you cannot open the tape device for writing. Closing the device has varying effects, depending on the device's minor-device number and whether the device was opened for reading or writing. If the tape had been read, the tape is rewound; if the no-rewind device was specified, the tape advances to the next file. In the case of writing, two tape marks are written at the current position and the tape is rewound; if the no-rewind device was specified, two tape marks are written and the tape is positioned between them. When you close a device that had been opened for writing, the tape volume ends at the current position; data beyond this point are undefined.

Hard errors may occur during tape operation. They include detecting the end-of-tape (EOT) reflector, reading an unexpectedly long record, or seeking a cooked tape into a tape mark. After an error, no further operations can be performed on the unit until the program closes the device and you rewind the tape. Soft parity errors may arise due to dirt on the tape, a bad tape, or misaligned heads. If an error occurs on a write, the device may attempt to place the record further along the tape. If the error occurs on a read, the driver simply rescans the record. After several failures, the driver announces a hard error.

Drivers

COHERENT includes two drivers for tape backups:

ft This driver has major number 4, the same as the floppy-disk drive. It works with QIC-40 and QIC-80 drives from Colorado, Archive, Mountain, Summit, and IBM.

hai This is a host adapter-independent SCSI driver, which supports SCSI hard disks as well as tape. This has major device number 13. **hai** works with hard disks from Adaptec, Seagate, and Future Domain. It has been tested with the Archive Viper 60, 150, 250, and 525 SCSI tape devices, and is known to work with them.

Each driver has a number of default behaviors, depending upon how you access it. For details, see the driver's entry in the Lexicon.

Devices

The following names the devices used to access tape drives. For SCSI tape devices, *N* is the SCSI identifier of your tape unit, as set when you installed COHERENT. (To change your suite of SCSI identifiers, you must reconfigure your kernel. For directions on how to do this, see the Lexicon entry for **hai**.)

/dev/rStp*N*	SCSI tape unit *N*, raw device, rewind.
/dev/nrStp*N*	SCSI tape unit *N*, raw device, no rewind.
/dev/xStp*N*	SCSI tape unit *N*, control device.
/dev/rct*N*	QIC-24 tape unit *N*, raw device, rewind.
/dev/nrct*N*	QIC-24 tape unit *N*, raw device, no rewind.
/dev/xct*N*	QIC-24 tape unit *N*, control device.
/dev/ft*N*	QIC-40/80 (floppy tape), rewind.
/dev/nft*N*	QIC-40/80 (floppy tape), no rewind.
/dev/ctmini	Default mini-cartridge device, retensioning.
/dev/rctmini	Default mini-cartridge device, no retensioning.
/dev/xctmini	Default mini-cartridge device, control device.
/dev/mc*N*	Irwin floppy tape, retensioning
/dev/rmc*N*	Irwin floppy tape, no retensioning.
/dev/xmc*N*	Irwin floppy tape, control device.

Installing Tape Devices

To install a SCSI tape device onto your system, do the following:

- Power down your system; then plug the SCSI device into your SCSI board. Do *not* plug the tape device into your SCSI board while your system is powered up, or you will damage your hardware.

- After you have rebooted your system, log in as the superuser **root**.
- **cd** to directory **/etc/conf**.
- Invoke the command **cohtune hai** and set the variable **HAI_TAPE** to the SCSI ID of the tape drive — usually two.
- Invoke the command **idmkcoh** to build a new kernel.
- Reboot your system and invoke the newly built kernel.

To install a floppy-tape device onto your system, do the following:

- If you have not already done so, make sure that the you have updated COHERENT to a version that supports floppy tape, that is, release 4.2.12 or later.
- Power down your system and install the floppy-tape device as described in its manual. Do not attempt to install your device while your system is powered up, or you can damage or destroy your system. Be very careful that DIP switches and jumpers are set correctly. Also, make sure that all cables are seated firmly — it is easy to loosen a connected while installing a tape device.
- Reboot your system into single-user mode. You can do so by typing **<ctrl-C>** while your file system is being checked during the reboot process, or invoke the command

  ```
  /etc/shutdown single 0
  ```

  after the system has checked its file system and rebooted.
- Running from single-user mode, run the script **/etc/conf/ft/mkdev**. If you know that your tape drive uses soft select and know the manufacturer, you can specify these features explicitly. If you know that your tape drive uses hard select and know the unit number (for example, a tape drive that takes the place of a second floppy-disk drive is unit 1), you can specify these features explicitly. If you are not sure of the above, select automatic configuration. The device driver **ft** will try to sense which type of drive you are using.
- Unless you have other tape drives installed, we recommend that you link the no-rewind-on-close floppy-tape device to the default tape device **/dev/tape**.
- While still in single-user mode, run the script **/etc/conf/bin/idmkcoh**. This generates a new kernel that can access the tape drive.
- Reboot your system and invoke the newly built kernel.

Manipulating Tape Devices

The command **tape** manipulates tape devices. With this command, you can rewind a tape, check the status of a tape device, or perform other useful tasks. For details, see its entry in the Lexicon.

Command **ftbad** lets you view and edit the list of bad blocks on a floppy-tape cartridge. For details on how to use this command, use see its entry in the Lexicon.

For details on how to build backups onto tape devices, see the Lexicon entry **backups**.

See Also

Administering COHERENT, backups, ft, ftbad, hai, tape [command]

Notes

Systems with a very slow CPU (e.g., a 16-megahertz 80386SX) may have trouble running the floppy-tape driver **ft** in multi-user mode. The reason is that floppy-tape hardware does not have much intelligence built into it, so the driver must consume many CPU cycles. In such instances, we suggest that you back up your system while in single-user mode (which is a good idea in any case).

tape — Command

Manipulate a tape device
tape command [*count***] [***device***]**

The command **tape** lets you manipulate a tape device. *device* names the tape device to manipulate. If you name no *device* on the command line, **tape** uses the device **T_DEFAULT**; header file **<tape.h>** defines this constant to be device **/dev/tape**. For a list of tape devices, see the overview article for **tape**.

command names the task that you want **tape** to execute, as follows:

erase Erase the tape. SCSI tape only.

retension Retension the tape. This rewinds the tape, then performs a full forward wind, then another rewind. The seek offset is set to zero.

rewind Rewind the tape. This command positions the tape at the beginning of track 0. It resets seek offset (see **seek** and **tell**, below) to zero. If tape is already rewound, this command has no effect.

rfm Move the tape forward to the next file mark; in effect, skip the current file. SCSI tape only.

seek *location* This command has the same effect as if the tape had just been used with no-rewind-on-close, leaving the tape at byte *location*. No tape motion occurs at the time of the command, but the next read or write begins at byte *location* on the tape. Floppy tape only.

status Display various parameters for the tape drive, and for the cartridge being used. Not every tape drive supports every status option. Unsupported features appear as "unavailable". The following gives an example of output from this command:

```
Floppy Tape Status:
      Drive Configuration  = 0x90
            500 Kbits/sec
            Non-Extra-Length Tape
            QIC-80 Mode.
      ROM Version  = 0x85
      Vendor ID  = 0x0146, Make=5, Model=6
      Tape Status Unavailable.
      Drive Status  = 0x65
            drive ready or idle
            cartridge present
            cartridge referenced
            at physical BOT
      Drive Error Status - No Error.
```

Floppy tape only.

tell Display the byte offset that will be in effect the next time the tape is read or written. Floppy tape only.

The related command **ftbad** lets you read and modify the list of bad blocks on a floppy-tape cartridge.

See Also

commands, ftbad, hai, tape

tar — Command

Archiving/backup utility
tar [*options***]** *file ...*

tar is a utility that lets you read, write, and update archives in a machine-independent format. Its name is an abbreviation for *tape archive;* however, **tar** can read/write output to files and floppy disks, as well as to magnetic tape.

tar is now a link to the command **gtar**, which implements tape archiving more robustly than did the version of **tar** shipped with earlier editions of COHERENT. For details on how to use **gtar**, turn to its entry in the Lexicon.

See Also

commands, gnucpio, gtar
POSIX Standard, §10.1.1

tboot — Technical Information

Describe the tertiary bootstrap

Booting is the process of loading COHERENT into memory and setting it into motion. This normally occurs after you have turned on your computer. The term comes from the old expression about pulling one's self up by one's bootstraps.

Booting can be quite involved, and uses a number of files, depending upon the version of COHERENT being booted and the medium from which you are booting it. The subject of this article, **tboot**, is the booting program that performs tertiary booting.

To grasp what is meant by "tertiary booting", consider how the boot sequence works:

1. The BIOS loads the first 512 bytes off of the first hard disk and runs it. This program is called the **master boot**. Mark Williams Company recommends that you use the COHERENT master boot, because it lets you boot off any partition on either of the first two drives.
2. The master boot loads the first 512 bytes off the active partition and runs that. This program is the "secondary boot" program.

   The secondary boot is generally responsible for loading the operating system off the active parition and running it.

Recent releases of COHERENT need a more sophisticated program to load the operating system than can fit into 512 bytes. In these releases of COHERENT, the secondary boot loads a program off the root file system; this program is called the "tertiary boot", or **tboot**.

tboot evaluates the hardware of your computer to provide the operating system (COHERENT) with vital information. This evaluation allows COHERENT to run without modification on a wider range of hardware.

tboot is responsible for loading the operating system kernel. It first looks for a file called **autoboot**, which it then loads. If **autoboot** does not exist, **tboot** prompts you to type in the name of a kernel, e.g., **begin** (during installation) or **coherent**. If you do not remember the name of the kernel you wish to boot, you can type **dir** or **ls** for a list of files in your root file system.

Pressing the spacebar when the prompt is displayed prevents execution of **/autoboot** and causes **tboot** to pause. You can then type the name of an alternate kernel to load (assuming it already resides within the root directory), type **ls** to see a listing of files, or type **info** for a display of hard-drive parameters.

See Also

Administering COHERENT, booting

tcdrain() — termios Macro (termios.h)

Drain output to a device
#include <termios.h>
int tcdrain(*fd***)**
int *fd***;**

The **termios** macro **tcdrain()** waits until all output written to device *fd* has been transmitted. *fd* must have returned by a call to **open()**, and must describe a terminal device.

If all goes well, **tcdrain()** returns zero. If something goes wrong, it returns -1 and sets **errno** to an appropriate value, as follows:

EBADF *fd* is not a valid file descriptor.

EINTR A signal interrupted **tcdrain()**.

ENOTTY
fd does not describe a terminal device.

See Also

termios
POSIX Standard, §7.2.2

tcflow() — termios Macro (termios.h)

Control flow on a terminal device
#include <termios.h>
int tcflow(*fd***,** *action***)**
int *fd***;**
int *action***;**

The **termios** macro **tcflow()** suspends transmission of data to, or receiption of data from, the device described by file descriptor *fd*. When a terminal device is opened, by default neither its input nor its output is suspended. *action* gives the action to take, as follows:

TCOOFF
Suspend output.

TCOON
Restart output.

TCIOFF
Transmit character **STOP**,
which tells the terminal to stop sending data to the system.

TCION Transmit character **START**, which tells the terminal to resume sending data to the system.

These constants are defined in header file **<termios.h>**.

Should all go well, **tcflow()** returns zero. If something goes wrong, it returns -1 and sets **errno** to an appropriate value, as follows:

EBADF *fd* is not a valid file descriptor.

EINVAL
action is not set to an appropriate value.

ENOTTY
fd does not describe a terminal device.

See Also

termios
POSIX Standard, §7.2.2

tcflush() — termios Macro (termios.h)

Flush data being exchanged with a terminal
#include <termios.h>
int tcflush (*fd***,** *queue_selector***)**
int *fd***;**
int *queue_selector***;**

The **termios** macro **tcflush()** discards, or "flushes," data send to or received from the terminal device described by the file descriptor *fd*. *queue_selector* indicates what to do, as follows:

TCIFLUSH
Flush data received but not read.

TCOFLUSH
Flush data written but not transmitted.

TCIOFLUSH
Flush both data written and data read.

These constants are defined in header file **<termios.h>**.

If all goes well, **tcflush()** returns zero. If something goes wrong, it returns -1 and sets **errno** to an appropriate value, as follows:

EBADF *fd* is not a valid file descriptor.

EINVAL
queue_selector is not a proper value.

ENOTTY
fd does not describe a terminal device.

See Also

termios
POSIX Standard, §7.2.2

tcgetattr() — termios Macro (termios.h)

Get terminal attributes
#include <termios.h>
int tcgetattr (*fd***,** *termios_p***);**
int fd;
struct termios *termios_p;

The **termios** macro **tcgetattr()** gets the parameters for the terminal device described by file descriptor *fd*, and stores them in the **termios** structure to which *termios_p* points.

ttcgetattr() can be called from a background process. Please note, however, a foreground process can subsequently change the terminal device's attributes, which renders obsolete the information in *termios_p*.

If all goes well, **tcgetattr()** returns zero. If a problem occurs, it returns -1 and sets **errno** to an appropriate value, as follows:

EBADF *fd* is not a valid file descriptor.

ENOTTY
fd does not describe a terminal device.

See Also

tcsetattr(), termios
POSIX Standard, §7.2.1

tcsendbreak() — termios Macro (termios.h)

Send a break to a terminal
#include <termios.h>
int tcsendbreak(*fd***,** *duration***)**
int *fd***;**
int *duration***;**

The **termios** macro **tcsendbreak()** transmits NUL characters to the terminal device described by file descriptor *fd*.

duration gives the length of time to transmit NUL characters. If *duration* is zero, **tcsnedbreak()** transmits zero-valued bits for at least 0.25 seconds and no more than 0.5 seconds. If *duration* is not set to zero, **tcsendbreak()** sends zero-valued bits for the time specified by the implementation. Under COHERENT, **tcsendbreak()** is a macro defined as follows:

```
#define tcsendbreak(filedes,duration) ioctl(filedes,TCSBRK,0)
```

TCSBRK is defined in header file **<termio.h>**: it transmits break characters for 0.25 seconds. Thus, the argument *duration* is ignored.

If *fd* does not use asynchronous serial data transmission, the implementation defines whether the **tcsendbreak()** function sends data to generate a break condition (as defined by the implementation) or returns without taking any action. Under COHERENT, it does nothing.

If all goes well, **tcsendbreak()** returns zero. If something goes wrong, it returns -1 and sets **errno** to an appropriate value, as follows:

EBADF *fd* is not a valid file descriptor.

ENOTTY
fd does not describe a terminal.

See Also

termios
POSIX Standard, §7.2.2

tcsetattr() — termios Macro (termios.h)

Set terminal attributes
#include <termios.h>
int tcsetattr(*fd***,** *optional_actions***,** *termios_p***)**
int *fd***,** *optional_actions***;**
struct termios **termios_p***;*

The **termios** macro **tcsetattr()** sets the attributes of the terminal device described by file descriptor *fd* to those stored in the **termios** structure to which *termios_p* points.

option_actions defines the manner in which the attributes are set, as follows:

TCSANOW
The attributes are set immediately.

TCSADRAIN
The attributes are set after all data that has been sent to *fd* has been written. Use this when changing parameters that affect output.

TCSAFLUSH
The change occurs after all output send to *fd* has been written: all input received but not read is discarded.

These constants are defined in header file **<termios.h>**.

If all goes well, **tcsetattr()** returns zero. If something goes wrong, it returns -1 and sets **errno**

EBADF *fd* is not a valid file descriptor.

EINVAL
optional_actions is not a proper value, or an attempt was made to change an attribute in the **termios** structure to an unsupported value.

ENOTTY
fd does not describe a terminal.

See Also

termios
POSIX Standard, §7.2.1

tee — Command

Copy input to multiple output streams
tee [-a] [-i] [*file ...***]**

tee reads from standard input, usually a pipe, and writes to the standard output, usually a pipe. **tee** also writes a copy of the input data to each *file* specified.

The **-a** flag tells **tee** to append data to each *file*, analogous to the shell construct "**>>***file*". Otherwise, it creates each *file,* analogous to the construct "**>***file*".

The flag **-i** means ignore interrupts.

See Also

commands, ksh, sh

telldir() — General Function (libc)

Return the current position within a directory stream
off_t telldir (*dirp***)**
DIR **dirp***;*

The COHERENT function **telldir()** is one of a set of COHERENT routines that manipulate directories in a device-independent manner. It returns the current position within the directory stream pointed to by *dirp*.

If an error occurs, **telldir()** exits and sets **errno** to an appropriate value.

See Also

closedir(), dirent.h, getdents(), libc, opendir(), readdir(), rewinddir(), seekdir(),

Notes

The value returned by **telldir()** should only be used as an argument to *seekdir()*.

telldir() and **seekdir()** are unreliable when directory stream has been closed and reopened. It is best to avoid using **telldir()** and **seekdir()** altogether.

Because directory entries can dynamically appear and disappear, and because directory contents are buffered by these routines, an application may need to continually rescan a directory to maintain an accurate picture of its active entries.

The COHERENT implementation of the **dirent** routines was written by D. Gwynn.

tempnam() — General Function (libc)

Generate a unique name for a temporary file
#include <stdio.h>
char ***tempnam(***directory***,** *name***);**
char **directory***,** **name***;**

tempnam() constructs a unique temporary name that can be used to name a file. *directory* points to the name of the directory in which you want the temporary file written. If this variable is NULL, **tempnam()** reads the environmental variable **TMPDIR** and uses it for *directory*. If neither *directory* nor **TMPDIR** is given, **tempnam()** uses **/tmp**.

name points to the string of letters that will prefix the temporary name. This string should not be more than three or four characters, to prevent truncation or duplication of temporary file names. If *name* is NULL, **tempnam()** sets it to **t**.

tempnam() uses **malloc()** to allocate a buffer for the temporary file name it returns. If all goes well, it returns a pointer to the temporary name it has written. Otherwise, it returns NULL if the allocation fails or if it cannot build a temporary file name successfully.

See Also

libc, mktemp(), TMPDIR, tmpfile(), tmpnam()

TERM — Environmental Variable

Name the default terminal type
TERM=*terminal type*

The environmental variable **TERM** names the type of terminal that you are using. This variable is read by every program that uses the **termcap** or **terminfo** library, to ensure that the correct terminal description is read when the program is invoked. You should set this variable in your **profile**, to ensure that the system understands what type of terminal you use. The file **/etc/profile** sets **TERM** to **ansipc**.

See Also

environmental variables, me, termcap

term — System Administration

Format of compiled terminfo file

Before it can be used, a file of **terminfo** information must be compiled with the command **tic**. It is read by the command **setupterm**.

Once compiled, the binary **terminfo** file is moved into a sub-directory of directory **/usr/lib/terminfo**. To avoid a linear search of a huge COHERENT directory, a two-level scheme is used to name the subdirectories: **/usr/lib/terminfo/***C***/***name*, where *name* names the terminal and *C* is the first character of *name*. For example, the **terminfo** entry for the Wyse 150 terminal is kept in the file **/usr/lib/terminfo/w/wyse150**. Synonyms for a terminal exist as links to the same compiled file.

The binary format of a **terminfo** file has been designed to be the same on all hardware. The file is divided into six parts: header, terminal names, boolean flags, numbers, strings, and string table.

Header

The *header* section begins the file. This section contains the following six short integers:

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.

A *short integer* is two bytes long. Under the **term** file format, 0xFFFF represents -1; all other negative value are illegal. Minus 1 generally means that a capability is missing from this terminal. All short integers are aligned on a short-word boundary.

Names

The *names* section contains the first line of the **terminfo** description, which lists the names for the terminal, each name separated by a vertical bar '|'. The section is terminated with a NUL.

Boolean

The *boolean* section contains the boolean flags for terminals. There is one flag for each boolean capacity recognized by **terminfo**. The flags appear in the order described in the header file **term.h**. Each flag is one byte long, and is set to zero or one, depending upon whether the capacity is absent or present in this terminal. If necessary, this section is ended with a NUL to ensure that the next section begins on an even byte.

Numbers

The *numbers* section is similar to the *flags* section. There is one entry for each numeric capacity recognized by **terminfo**, each capacity being represented by a short integer. A value of -1 indicates that this terminal lacks this capability. Entries appear in the order described in the header file **term.h**.

Strings

The *strings* section also contains one short integer for each string capability recognized by **terminfo**. A value of -1 means that this terminal lacks this capability. Otherwise, the value gives an offset from the beginning of the string table. Entries appear in the order described in the header file **term.h**.

Special characters in **^X** or *****c notation are stored in their interpreted form. Padding information and parameter information are stored intact in uninterpreted form.

String Table

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.

Files

/usr/lib/terminfo/* — Default location of object files

See Also

Administering COHERENT, curses, infocmp, tic, terminfo
Strang, J., Mui, L., O'Reilly, T.: *termcap and terminfo*. Sebastopol, CA: O'Reilly & Associates, Inc., 1991.

Notes

The total compiled file cannot exceed 4,096 bytes. The *name* field cannot exceed 128 bytes.

termcap — System Administration

Terminal-description language
/etc/termcap

termcap is a language for describing terminals and their capabilities. Terminal descriptions are collected in the file **/etc/termcap** and are read by **tgetent** and its related programs to ensure that output to a particular terminal is in a format that that terminal can understand.

COHERENT also supports the terminal-description language **terminfo**. For a description of how these languages differ, see the Lexicon entry for **terminfo**.

Overview

A terminal description written in **termcap** consists of a series of fields, which are separated from each other by colons ':'. Every line in the description, with the exception of the last line, must end in a backslash '\'. Tab characters are ignored. Lines that begin with a '#' are comments. A **termcap** description must not exceed 1,024 characters.

The first field names the terminal. Several different names may be used, each separated by a vertical bar '|'; each name given, however, must be unique within the file **/etc/termcap**. By convention, the first listed must be two characters long. The second name is the name by which the terminal is most commonly known; this name may contain no blanks in it. Other versions of the name may follow. By convention, the last version is the full name of the terminal; here, spaces may be used for legibility. Any of these may be used to name your terminal to the COHERENT system. For example, the name field for the VT-100 terminal is as follows:

```
d1|vt100|vt-100|pt100|pt-100|dec vt100:\
```

Note that the names are separated by vertical bars '|', that the field ends with a colon, and that the line ends with a backslash. Using any of these names in an **export** command will make the correct terminal description available to programs that need to use it.

The remaining fields in the entry describe the *capabilities* of the terminal. Each capability field consists of a two-letter code, and may include additional information. There are three types of capability:

Boolean

This indicates whether or not a terminal has a specific feature. If the field is present, the terminal is assumed to have the feature; if it is absend, the terminal is assumed not to have that feature. For example, the field

am:

is present, **termcap** assumes that the terminal has automatic margins, whereas if that field is not present, the program using **termcap** assumes that the terminal does not have them.

Numeric

This gives the size of some aspect of the terminal. Numeric capability fields have the capability code, followed by a '#' and a number. For example, the entry

co#80:

means that the terminal screen is 80 columns wide.

String capabilities

These give a sequence of characters that trigger a terminal operation. These fields consist of the capability code, an equal sign '=', and the string.

Strings often include escape sequences. A "**\E**" indicates an **<ESC>** character; a control character is indicated with a carat '^' plus the appropriate letter; and the sequences **\b, \f, \n, \r**, and **\t** are, respectively, backspace, formfeed, newline, **<return>**, and tab.

An integer or an integer followed by an asterisk in the string (e.g., '*int**') indicates that execution of the function should be delayed by *int* milliseconds; this delay is termed *padding*. Thus, deletion on lines on the Microterm Mime-2A is coded as:

```
dl=20*^W:
```

dl is the capability code for *delete*, the equal sign introduces the deletion sequence, **20*** indicates that each line deletion should be delayed by 20 milliseconds, and **^W** indicates that the line-deletion code on the Mime-2A is **<ctrl-W>**.

The asterisk indicates that the padding required is proportional to the number of lines affected by the operation. In the above example, the deletion of four lines on the Mime-2A generates a total of 80 milliseconds of padding; if no asterisk were present, however, the padding would be only 20 milliseconds, no matter how many lines were deleted. Also, when an asterisk is used, the number may be given to one decimal place, to show tenths of a millisecond of padding.

Note that with string capabilities, characters may be given as a backslash followed by the three octal digits of the character's ASCII code. Thus, a colon in a capability field may be given by **\072**. To put a null character into the string, use **\200**, because **termcap** strips the high bit from each character.

LEXICON

Finally, the literal characters '**^**' and '****' are given by "**\^**" and "****".

Capability Codes

The following table lists **termcap**'s capability codes. **Type** indicates whether the code is boolean, numeric, or string; a dagger '†' indicates that this capability may include padding, and a dagger plus an asterisk "†*" indicates that it may be used with the asterisk padding function described above.

Variable	*Type*	*Definition*
ae	string†	End alternate set of characters
al	string†*	Add blank line
am	boolean	Automatic margins
as	string†	Start alternate set of characters
bc	string	Backspace character, if not **<ctrl-H>**
bs	boolean	Backspace character is **<ctrl-H>**
bt	string†	Backtab
bw	boolean	Backspace wraps from column 0 to last column
CC	string	Command character in prototype if it can be set at terminal
cd	string†*	Clear to end of display
ce	string†	Clear line
ch	string†	Horizontal cursor motion
cl	string†*	Clear screen
cm	string†	Cursor motion, both vertical and horizontal
co	number	Number of columns
cr	string†*	**<return>**; default **<ctrl-M>**
cs	string†	Change scrolling region (DEC VT100 only); resembles **cm**
cv	string†	Vertical cursor motion
da	boolean†	Display above may be retained
dB	number	Milliseconds of delay needed by **bs**
db	boolean	Display below may be retained
dC	number	Milliseconds of delay needed by **cr**
dc	string†*	Delete a character
dF	number	Milliseconds of delay needed by **ff**
dl	string†*	Delete a line
dm	string	Enter delete mode
dN	number	Milliseconds of delay needed by **nl**
do	string	Move down one line
dT	number	Milliseconds of delay needed by tab
ed	string	Leave delete mode
ei	string	Leave insert mode; use **:ei=:** if this string is the same as **ic**
eo	string	Erase overstrikes with a blank
ff	string†*	Eject hardcopy terminal page; default **<ctrl-L>**
hc	boolean	Hardcopy terminal
hd	string	Move half-line down, i.e., forward 1/2 line feed)
ho	string	Move cursor to home position; use if **cm** is not set
hu	string	Move half-line up, i.e., reverse 1/2 line feed
hz	string	Cannot print tilde '~' (Hazeltine terminals only)
ic	string†	Insert a character
if	string	Name of the file that contains **is**
im	string	Begin insert mode; use **:im=:** if **ic** has not been set
in	boolean	Nulls are distinguished in display
ip	string†*	Insert padding after each character listed
is	string	Initialize terminal
k0-k9	string	Codes sent by function keys 1 through 10 (**k0** = F10)
kb	string	Code sent by backspace key
kd	string	Code sent by down-arrow key
ke	string	Leave "keypad transmit" mode
kh	string	Code sent by home key
kl	string	Code sent by left-arrow key
kn	number	No. of function keys; default is 10
ko	string	Entries for for all other non-function keys
kr	string	Code sent by right-arrow key
ks	string	Begin "keypad transmit" mode

ku . . . . . string . . . . . Code sent by up-arrow key
l0-l9. . . . . string . . . . . Function keys labels if not f0-f9
li. . . . . number . . . . . Number of lines
ll. . . . . string . . . . . Move cursor to first column of last line (**cm** not set)
ma. . . . . string . . . . . Map keypad-to-cursor movement for **vi** version 2
mi . . . . . boolean . . . . . Cursor may be safely moved while in insert mode
ml . . . . . string . . . . . Turn on memory lock for area of screen above cursor
ms. . . . . boolean . . . . . Cursor can be moved while in standout or underline mode
mu . . . . . string . . . . . Turn off memory lock
nc . . . . . boolean . . . . . **<return>** does not work
nd . . . . . string . . . . . Move cursor right non-destructively
nl . . . . . string†* . . . . . Newline character; default is **\n** (*Obsolete*)
ns . . . . . boolean . . . . . Terminal is CRT, but does not scroll
os . . . . . boolean . . . . . Terminal can overstrike
pc . . . . . string . . . . . Pad character any character other than null
PS . . . . . string . . . . . Print start: redirect input to auxiliary port
PN . . . . . string . . . . . Print end: stop redirecting input to auxiliary port
pt . . . . . boolean . . . . . Terminal's tabs set by hardware; may need to be set with **is**
se . . . . . string . . . . . Exit standout mode
sf . . . . . string† . . . . . Scroll forward
sg . . . . . number . . . . . Blank characters left by **so** or **se**
so . . . . . string . . . . . Enter standout mode
sr . . . . . string† . . . . . Reverse scroll
ta . . . . . string† . . . . . Tab character other than **<ctrl-I>**, or with padding
tc . . . . . string . . . . . Similar terminal — must be last field in entry
te . . . . . string . . . . . End a program that uses **cm**
ti . . . . . string . . . . . Begin a program that uses **cm**
uc . . . . . string . . . . . Underscore character and skip it
ue . . . . . string . . . . . Leave underscore mode
ug . . . . . number . . . . . Blank characters left by **us** or **ue**
ul . . . . . boolean . . . . . Terminal underlines but does not overstrike
up . . . . . string . . . . . Move up one line
us . . . . . string . . . . . Begin underscore mode
vb . . . . . string . . . . . Visible bell; may not move cursor
ve . . . . . string . . . . . Exit open/visual mode
vs . . . . . string . . . . . Begin open/visual mode
xb . . . . . boolean . . . . . Beehive terminal (f1=**<esc>**, f2=**<crtl-C>**)
xn . . . . . boolean . . . . . Newline is ignored after wrap
xr . . . . . boolean . . . . . **<return>** behaves like **ce \r \n**
xs . . . . . boolean . . . . . Standout mode is not erased by writing over it
xt . . . . . boolean . . . . . Tabs are destructive

Examples

The following is the **termcap** description for the IBM Personal Computer, also known as **ansipc**. This is the default description used with your COHERENT system console:

```
ap|ansipc|ansi personal computer:\
        :al=\E[L:am:bs:bt=\E[Z:bw:cd=\E[O:ce=\E[K:ch=\E[%i%d‘:\
        :cl=\E[2O:cm=\E[%i%d;%dH:co#80:cs=\E[%i%d;%dr:\
        :cv=\E[%i%dd:dl=\E[M:ho=\E[H:is=\E[25f\E[2K\E[m\E[H:\
        :k0=\E[0x:k1=\E[1x:k2=\E[2x:k3=\E[3x:k4=\E[4x:k5=\E[5x:\
        :k6=\E[6x:k7=\E[7x:k8=\E[8x:k9=\E[9x:kb=^h:kd=\E[B:kh=\E[H:\
        :kl=\E[D:kr=\E[C:ku=\E[A:li#24:ll=\E[24;1H:hd=\E[C:se=\E[m:\
        :sf=\E[S:sg#0:so=\E[7m:sr=\E[T:ue=\E[m:up=\E[A:us=\E[4m:\
        :KI=\E[5x:KD=\E[3x:Kd=\E[P:KB=\E[6x:KU=\E[4x:Ku=\E[@:\
        :KM=\E[7x:KJ=\E[8x:Kt=\E[Z:KT=\t:KL=\E[1x:KR=\E[2x:KP=\E[U:\
        :Kp=\E[V:KX=\E[9x:KC=\E[0x:KE=\E[24H:KW=^F:Kw=^R:Kr=^N:do=\E[B:
```

The first field, which occupies line 1, gives the various aliases for this device. The remaining fields mean the following:

:al=\E[L: **<esc>L** adds new blank line; use one millisecond for each line added.

:am: Terminal has automatic margins.
:bs: Backspace character is **<ctrl>-H** (the default).
:bt=\E[Z: **<esc>[Z** back-tabs.
:bw: On this device, a backspace character wraps from column 0 to the last column (in this case, column 79) on the previous line.
:cd=\E[O: **<esc>[O** clears to the end of display.
:ce=\E[K: **<esc>[K** clears to end of line.
:ch=\E[%i%d‘: The string for horizontal cursor motion (described later).
:cl=\E[2O: **<esc>[2O** clears screen.
:cm=\E[%i%d;%dH:
Cursor motion (described later).
:co#80: Screen has 80 columns.
:cs=\E[%i%d;%dr:
String for changing the scrolling region.
:cv=\E[%i%dd: String for vertical cursor motion.
:dl=\E[M: **<esc>E[M** deletes a line.
:ho=\E[H: **<esc>[H** moves cursor to home position.
:is=\E[25f\E[2K\E[m\E[H:
The string with which the device is initialized.
:k0=\E[0x: Function key 10 sends sequence **<esc>[0x**.
:k1=\E[1x: Function key 1 sends sequence **<esc>[1x**.
:k2=\E[2x: Function key 2 sends sequence **<esc>[2x**.
:k3=\E[3x: Function key 3 sends sequence **<esc>[3x**.
:k4=\E[4x: Function key 4 sends sequence **<esc>[4x**.
:k5=\E[5x: Function key 5 sends sequence **<esc>[5x**.
:k6=\E[6x: Function key 6 sends sequence **<esc>[6x**.
:k7=\E[7x: Function key 7 sends sequence **<esc>[7x**.
:k8=\E[8x: Function key 8 sends sequence **<esc>[8x**.
:k9=\E[9x: Function key 9 sends sequence **<esc>[9x**.
:kb=^h: Backspace key sends **<Ctrl>-H**.
:kd=\E[B: Down-arrow key sends **<esc>[B**.
:kh=\E[H: Home key sends **<esc>[H**.
:kl=\E[D: Left-arrow key sends **<esc>[D**.
:kr=\E[C: Right-arrow key sends **<esc>[C**.
:ku=\E[A: Up-arrow key sends **<esc>[A**.
:li#24: Terminal has 24 lines.
:ll=\E[24;1H: **<esc>[24;1H** moves the cursor to the first column of the last line.
:hd=\E[C: **<esc>[C** moves the cursor downward by one-half line.
:se=\E[m: **<esc>[m** exits standout mode.
:sf=\E[S: **<esc>[S** scrolls the screen forward.
:sg#0: The **so** and **se** instructions leave zero blank lines on the screen.
:so=\E[7m: **<esc>[7m** begins standout mode.
:sr=\E[T: **<esc>[T** reverse-scrolls the screen.
:ue=\E[m: **<esc>m** ends underline mode.
:up=\E[A: **<esc>[A** moves the cursor up one line.
:us=\E[4m: **<esc>4m** begins underscore mode.
:do=\E[B: **<esc>E[B** moves cursor down one line.

Note that the last field did *not* end with a backslash; this indicated to the COHERENT system that the **termcap** description was finished.

A terminal description does not have to be nearly so detailed. If you wish to use a new terminal, first check the following table to see if it already appears by **termcap**. If it does not, check the terminal's documentation to see if it mimics a terminal that is already in **/etc/termcap**, and use that description, modifying it if necessary and changing the name to suit your terminal. If you must create an entirely new description, first prepare a skeleton file that contains the following basic elements: number of lines, number of columns, backspace, cursor motion, line delete, clear screen, move cursor to home position, newline, move cursor up a line, and non-destructive right space. For example, the following is the **termcap** description for the Lear-Siegler ADM-3A terminal:

```
la|adm3a|3a|lsi adm3a:\
        :am:bs:cd=^W:ce=^X:cm=\E=%+ %+ :cl=^Z:co#80:ho=^^:li#24:\
        :nd=<ctrl-L>:up=^K:
```

Once you have installed and debugged the skeleton description, add details gradually until every feature of the terminal is described.

Cursor Motion

The cursor motion characteristic contains **printf**-like escape sequences not used elsewhere. These encode the line and column positions of the cursor, whereas other characters are passed unchanged. If the **cm** string is considered as a function, then its arguments are the line and the column to which the cursor is to move; the **%** codes have the following meanings:

%d Decimal number, as in **printf**. The origin is 0.

%2 Two-digit decimal number. The same as **%2d** in **printf()**.

%3 Three-digit decimal number. The same as **%3d** in **printf()**.

%. Single byte. The same as **%c** in **printf()**.

%+*n* Add *n* to the current position value. *n* may be either a number or a character.

%>*nm* If the current position value is greater than *n*+*m*; then there is no output.

%r Reverse order of line and column, giving column first and then line. No output.

%i Increment line and column.

%% Give a **%** sign in the string.

%n Exclusive or line and column with 0140 (Datamedia 2500 terminal only).

%B Binary coded decimal (16 * (*n*/10))+(*n*%10). No output.

%D Reverse coding (*n*-**(2*(***n***%16)**). No output (Delta Data terminal only).

To send the cursor to line 3, column 12 on the Hewlett-Packard 2645, the terminal must be sent **<esc>&a12c03Y** padded for 6 milliseconds. Note that the column is given first and then the line, and that the line and column are given as two digits each. Thus, the **cm** capability for the Hewlett-Packard 2645 is given by:

```
:cm=6\E&%r%2c%2Y:
```

The Microterm ACT-IV needs the current position sent preceded by a **<Ctrl-T>**, with the line and column encoded in binary:

```
:cm=^T%.%.:
```

Terminals that use **%.** must be able to backspace the cursor (**bs** or **bc**) and to move the cursor up one line on the screen (**up**). This is because transmitting **\t**, **\n**, **\r**, or **<ctrl-D>** may have undesirable consequences or be ignored by the system.

Similar Terminals

If your system uses two similar terminals, one can be defined as resembling the other, with certain exceptions. The code **tc** names the similar terminal. This field must be *last* in the **termcap** entry, and the combined length of the two entries cannot exceed 1,024 characters. Capabilities given first over-ride those in the similar terminal, and capabilities in the similar terminal can be cancelled by **xx@** where *xx* is the capability. For example, the entry

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

defines a Hewlett-Packard 2621 terminal that does not have the **ks** and **ke** capabilities, and thus cannot turn on the function keys when in visual mode.

Initialization

A terminal initialization string may be given with the **is** capability; if the string is too long, it may be read from a file given by the **if** code. Usually, these strings set the tabs on a terminal with settable tabs. If both **is** and **if** are given, **is** will be printed first to clear the tabs, then the tabs will be set from the file specified by **if**. The Hewlett-Packard 2626 has:

```
:is=\E&j@\r\E3\r:if=/usr/lib/tabset/stdcrt:
```

Programming With termcap

The COHERENT library **libterm.a** contains the following routines that extract and use the descriptions for **termcap**:

LEXICON

tgetent() Read a **termcap** entry.

tgetflag() Check if a given Boolean capability is present in the terminal's **termcap** entry.

tgetnum() Return the value of a numeric **termcap** feature (e.g., the number of columns on the terminal).

tgetstr() Read and decode a **termcap** string feature.

tgoto() Read and decode a cursor-addressing string.

tputs() Read and decode the leading padding information of a **termcap** string feature.

See the Lexicon entry for each function for details.

The external variable **ospeed** is the output speed to the terminal as encoded by **stty**. The external variable **PC** is a padding character if a NUL (**<crtl-@>**) is not appropriate.

The following example shows how to read a **termcap** entry:

```
#include <stdio.h>

static char *CM, *SO, *SE, *CL;
static int rows, cols;
static int am;
static int errflag;
static char *ptr;
static char *tv_stype;

extern char *tgoto();                       /* termcap cursor position command */
extern char *tgetstr();                     /* get string code from termcap */
extern int tgetflag();                      /* get boolean flag from termcap */
extern int tgetnum();                       /* get numeric code from termcap */
extern void tputs();                        /* termcap put data command */
extern char PC;                             /* termcap's pad character */

/*
 * Get a required termcap string or exit with a message.
 */
static char *
qgetstr(ref)
char *ref;
{
      register char *tmp;

      if ((tmp = tgetstr(ref, &ptr)) == NULL) {
            printf("/etc/termcap terminal %s must have a %s= entry\n",
                  tv_stype, ref);
            errflag = 1;
      }
      return (tmp);
}

/*
 * Get required termcap information for this terminal type.
 */
static void
tcapopen()
{
      extern char *getenv(), *realloc();
      char *tcapbuf;
      char tcbuf[1024]; /* this must hold the whole tml entry */
      char *p;

      /* set up termcap type */
      if ((tv_stype = getenv("TERM")) == NULL) {
            printf("Environment variable TERM not defined\n");
            exit(1);
      }
```

```
      if (tgetent(tcbuf, tv_stype) != 1) {
            printf("Terminal type %s not in /etc/termcap\n", tv_stype);
            exit(1);
      }

      /* get far too much and shrink later */
      if ((ptr = tcapbuf = malloc(1024)) == NULL) {
            printf("out of space\n");
            exit(1);
      }

      /* get termcap entries for later use */
      CM = qgetstr("cm"); /* this string used by tgoto() */
      CL = qgetstr("cl"); /* this string used to clear screen */
      SO = qgetstr("so"); /* this string used to set standout */
      SE = qgetstr("se"); /* this string used by clear standout */
      if (errflag)      /* set if any missing entries */
            exit(1);

      /* set termcap's pad char */
      PC = (((p = tgetstr("pc", &ptr)) == NULL) ? 0 : *p);

      if (tcapbuf != realloc(tcapbuf, (unsigned)(ptr - tcapbuf))) {
            printf("Buffer not shrunk in place!\n");
            exit(1);
      }

      if ((cols = tgetnum("co")) < 0)     /* Get rows and columns */
            cols = 80;
      if ((rows = tgetnum("li")) < 0)
            rows = 24;

      am = tgetflag("am");    /* automatic margins ? */
}

/*
 * output char function.
 */
static void
ttputc(c)
{
      fputc(c, stdout);
}

/*
 * output command string, set padding to one line affected.
 * use ttputc as character output function. Use only for
 * termcap created data not your own strings.
 */
void
putpad(str)
char *str;
{
      tputs(str, 1, ttputc);
}

/*
 * Move cursor.
 */
void
move(col, row)
{
      putpad(tgoto(CM, col, row));
}
```

```
/*
 * Demonstrate termcap.
 */
main()
{
      tcapopen();

      putpad(CL);             /* clear the screen */

      move(30, 5);
      putpad(SO);             /* standout mode */
      printf("Termcap Demo");
      putpad(SE);             /* end standout mode */

      move(0, 7);
      printf("This terminal has %d columns and %d rows.", cols, rows);

      if (am) {
            move(0, 8);
            printf("Automatic margins.");
      }

      move(0, rows);    /* quit at bottom of screen */
      exit(0);
}
```

Files

/etc/termcap — Terminal-description data base
/usr/lib/libterm.a — Routines for reading a **termcap** description

See Also

Administering COHERENT, captoinfo, curses, libterm, terminal, terminfo, tgetent(), tgetflag(), tgetnum(), tgetstr(), tgoto(), tputs()
Strang, J., Mui, L., O'Reilly, T.: *Termcap & Terminfo.* Sebastopol, CA: O'Reilly & Associates, Inc., 1991. *Highly recommended.*

Notes

To see which terminals are currently supported, see file **/etc/termcap**.

COHERENT also supports **terminfo**, a clone of the UNIX System-V terminal-description system. **terminfo** enjoys a number of features not available under **termcap**, and is the preferred system under COHERENT.

Should you wish to convert to **terminfo**, the command **captoinfo** converts a file of **termcap** descriptions to their **terminfo** analogues.

terminal — Technical Information

This article describes how you can hook up a terminal to your COHERENT system via a serial port. It also discusses common problems that arise with this procedure, as diagnosed daily by the technical support staff at Mark Williams Company. For information on connecting a modem to your computer's serial port, see the article **modem**.

Terminals for Beginners

To a beginner, a terminal — with its many wires and controls — may be daunting. However, connecting a terminal or PC-based terminal emulator to a COHERENT system is really very easy. To make matters even easier, this section discusses how to use a simple configuration, using only one COM port.

What you need:

1. A COHERENT system with a COM port.
2. Either a terminal or a PC with a COM port and communications software. An old PC or PC-AT is fine. (You can also use a Macintosh or other such computer, but that is beyond the scope of this discussion.)
3. A "null modem cable" purchased from your nearest computer supply store. This should cost between $10 and $20, in U.S. currency. The only tricky part about the cable is making sure that the connectors on the ends match the connectors on your COM ports (i.e., nine pin vs. 25 pin, and male vs female. Adaptors are also available.

What you do:

1. Connect the cable to the COM port on the COHERENT system. Note that the COM port on your COHERENT system is always a *male* plug (that is, it has pins rather than sockets). Do *not* plug your connection into a female plug, as this is the parallel port. If you do so, you can damage your equipment.

2. Connect the cable to the COM port on the terminal/PC. If you are using a PC as a terminal, the COM port on it will also be male. If you are using a stand-alone terminal, the plug could be either male or female.

   If you are using a stand-alone terminal, there may be a plug on the back that is labelled "AUX". Do *not* use this plug; use the other one.

3. Configure the terminal/PC to use 9600 speed (or "baud") and 8N1 character setting (that is, eight bits, no parity, one stop bit. Note that you do not need a telephone number: you won't be dialing anywhere.

4. Log in on your COHERENT system as the superuser **root**. Type the following command:

   ```
   /etc/enable com?l
   ```

   where '?' is the number of the COM port on your COHERENT system into which you are plugging the cable from the PC or terminal Note that the last character is lower-case 'L', *not* a one.

5. Now come a tricky part: use your favorite text editor and edit file **/etc/ttytype**. This file gives the default type of terminal that will be connecting to COHERENT via a given COM port. This is important. If you don't do this, such screen-oriented programs as editors and **vsh** will not work properly.

   Each entry in **/etc/ttytype** has two columns: the first gives a type of terminal, and the second names the port. The following shows an example entry:

   ```
   vt100 com3l
   ```

   In this instance, COM port **com3l** has a DEC VT-100 plugged into it by default. Look for an entry for the COM port into which you plugged your terminal device. If you can't find you, you can create one easily enough; however, having two entries for the same port is not a good idea, as COHERENT will become confused.

   If you are plugging in a PC, use the terminal type **vt100**. If you are using a stand-alone terminal, name the type of terminal you are using, e.g., **wyse370** for a Wyse 370 color terminal. If you cannot figure out what type of terminal you are using, use **dumb**. This will not allow you to use screen-oriented programs like MicroEMACS, but at least you will be able to connect to COHERENT; you can figure out the type of terminal later.

6. Return to your terminal device. If you are using a PC, invoke the terminal emulator, and put it into "connect" mode. If you are using a stand-alone terminal, turn it on. Press `(¢)`. After a short pause, you should see the prompt

   ```
   Coherent login:
   ```

   on your screen. You can then log in and run normally.

That's all there is to it. In this way, you can get more use out of an old, obsolete PC. After you get it working, you may need to adjust some other settings, particularly if you are using a communications package on a PC for a terminal.

If you see garbage characters when you log in, probably the speed (or "baud rate") is not set correctly either on the COM port of your COHERENT system or on your terminal device. The Lexicon article on **ttys** describes the magic character string used to describe each com port and one of the letters is the port speed.

Problems with Terminal Hookup

As noted above, it is easy to hook up a terminal. However, problems can arise if you overlook any details.

When problems arise, they usually come from some form of confusion. These can include send/receive confusion, baud rate confusion, and shell/no shell confusion. The following sections describes eacy type of confusion in turn.

Send/Receive Confusion

This most often happens when you've soldered your own connectors, and you made a mistake. If you purchased a connector, as we recommended above, then skip to the next section.

A serial connection between your computer and a terminal requires at least three wires: one each for pins 2, 3, and 7. These pins, respectively, control send (TD), receive (RD), and signal-ground (Gnd or SG). These pin numbers

correspond to the 25-pin "DB-25" connectors used on most equipment. If your system has the AT-style nine-pin "DB-9" connectors, you will need to wire to the corresponding signals. See the Lexicon entry for **RS-232** for details of the pin-outs for these two connectors.

When hooking up a terminal to a serial port using a three-wire connection, you must cross pins 2 and 3, so that each device's send pin talks to the other device's receive pin. You can plug a device called a "null modem" between the cable and the serial port, to do this automatically.

Note that the only symptom of a problem in the cable is that nothing appears on your terminal when you type.

Baud-Rate Confusion

The terminal and the computer must speak to each other at the same speed, or "baud rate". A typical symptom of baud-rate confusion is garbage characters on the screen. When the wiring is wrong, you see nothing; when the baud rate is wrong, you see random collections of characters on the screen, but nothing meaningful.

You can fix baud-rate problems by using the command **stty** to reset the baud rate on the port, or resetting the baud rate on the terminal. The problem should also be solved by editing file **/etc/ttys**. For directions on how to reset the baud rate for a port, see the Lexicon entry for **stty**; see the Lexicon entry for **ttys** for information on how to edit **/etc/ttys**.

Please note, too, that COHERENT supports the following configuration for terminals:

8 word bits
1 stop bit
No parity bits

These settings, as well as the baud rate, must match before your terminal will work correctly.

The Old Shell Game

Before a terminal is useful to you, you must *enable* the port into which it is plugged. Enabling a port means that the COHERENT system creates a shell for that port: this, in turn, means that COHERENT prints a login prompt on the device plugged into that port, and reads and processes interactively commands that are entered from that port. The COHERENT system also restricts permissions on all enabled serial ports, so that only the superuser **root** can read and write to the port. This prevents other users who may be using the system from accessing the serial port.

Note that not all ports need be enabled: for example, you should not enable a COM port into which you've plugged a printer.

When you boot the COHERENT system, it reads system file **/etc/ttys** and creates a shell for each serial port that needs one. One way to enable a port is to log in as the superuser **root**, then use a text editor to change the port's entry in **/etc/ttys**, as described its Lexicon article. Finally, typing the command

```
kill quit 1
```

forces COHERENT to re-read **/etc/ttys** and so create a shell for the port. Note that doing this will ensure that the port is re-enabled every time you boot.

A better way to enable a port is to use the command **enable**, as described in its Lexicon article. For example, to put up a shell on COM port **/dev/com1r**, log in as the superuser **root** and type the command:

```
/etc/enable com1r
```

Exiting Raw Mode

A terminal is in *cooked* mode. In cooked mode, the tty driver interprets and correctly processes such predefined characters as the end-of-file character or the quit character. In *raw* mode, however, processing of such characters is turned off; and in general the terminal will behave bizarrely. Raw mode is used by programs that do not want the tty driver to interpret characters; for example, a program that uses a tty to transmit a binary to another machine does not want the tty driver to be interpreting the binary information being passed through it.

Occasionally, a program will exit abruptly and leave the terminal in raw mode. To return to cooked mode, use the command **<ctrl-J> stty sane <ctrl-J>**. This invokes the command **stty**, which lets you manipulate terminal settings, to restore the previous cooked state. See the Lexicon entry on **stty** for details on raw and cooked modes; this article also describes the options of this most useful command.

See Also

Administering COHERENT, asy, device drivers, hs, modem, RS-232, sgtty, stty, termcap, terminfo, termio, termios, ttys

Notes

One final bit of hard-won wisdom: once you have something working, write down what you did, and store it in a place where you won't lose it. Note especially what connectors are where and how they have been cabled together. It makes life easier just knowing that you are looking for a female-to-female cable instead of male-to-female or male-to-male. If you know whether to insert a null modem, you are even better off.

COHERENT supports multi-port serial cards as well as COM ports 1 through 4. See the Lexicon entry on **asy** for a list of the devices that COHERENT supports, and for details.

Thanks to Dave Hough (tecdah1@sdc.cs.boeing.com), whose posting to comp.os.coherent is the basis of the "Beginners" section, above.

terminfo — System Administration

Terminal-description language
/usr/lib/terminfo

terminfo is a system for describing terminals. Descriptions are collected in the file **/usr/lib/terminfo** and are read by **curses**, **more**, **vi**, and other utilities. By passing her terminal's **terminfo** entry to a program, a user can make sure that the program can take full advantage of her terminal's capacities.

terminfo resembles the terminal-description language **termcap**; however, it enjoys a number of features that **termcap** does not, as follows:

- A **termcap** entry cannot exceed a predefined limit. **terminfo** lifts this restriction.
- **terminfo** entries are compiled; therefore, they are read and loaded more quickly.
- **termcap** entries are all kept in file **/etc/termcap**. Each **terminfo** resides in its own file; thus, a program and find and load an entry more quickly.
- **terminfo** is a little more easily read by human beings.

Whether a program uses **termcap** or **terminfo** descriptions depends entirely on that program. For example, MicroEMACS uses **termcap** descriptions; but **vsh** (and other **curses**-based programs) use **terminfo**. In general, **terminfo** is regarded as being more flexible and up-to-date.

terminfo Entries

Directory **/usr/lib/terminfo** consists of a number of sub-directories, one for each terminal type being described. A *terminal type* describes a given make of terminal (e.g., the Wyse 150) plus some special attribute, such as the number of characters on a line or a specially defined bank of function keys. A **terminfo** entry can extend over more than one line by indenting every line after the first. A line that begins with a pound sign '#' is a comment.

A **terminfo** entry consists of an indefinite number of comma-separated fields. White space after each comma is ignored. The first field names the terminal; the remaining fields hold capability codes. (*Capability codes* are discussed in detail below.) Preceding a field with a period '.' comments out that field, and only that field.

Naming Terminals

The first field in a **terminfo** entry names the terminal being described. The name field consists of one or more names, which are separated by vertical-bar characters. The first name given is the most common abbreviation for the terminal. The last name is usually a long name that fully identifies the terminal. All names in between the first and the last give common synonyms for that terminal. All names can contain upper-case characters; the last name can also contain white space.

Terminal names (except for the last, verbose entry) should use the following conventions:

- The hardware should have a root name chosen, e.g., "wyse150".
- The root name should not contain hyphens, except to prevent synonyms from colliding 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. For example, a **wyse150** with an old-fashioned 82-key keyboard could be called **wyse150-o**.

Use the following suffixes whenever possible:

Suffix	*Meaning*
-w	Wide (more than 80 columns)
-am	With automatic margins (usually default)
-nam	Without automatic margins
-*n*	Number of lines on the screen
-na	No arrow keys
-*n***p**	*n* pages of memory
-rv	Reverse video

Capability Codes

A *capability code* describes a capability of a terminal. Capability codes come in three varieties:

Boolean This indicates whether a terminal has a given feature. If the field is present, the terminal is assumed to have the capability; if not, then it is assumed not to be present. For example, the code **am** indicates "automatic margins". If **am** appears in a terminal's **terminfo** entry, then it can execute automatic margins; if not, then it can't.

Numeric This gives the size of some aspect of a terminal, such as the number of lines or the number of columns. A numeric code is followed by a number sign '#' and then a string of digits, which set the value for that code For example, the code **cols#80** indicates that a terminal has 80 columns per row.

String Capabilities
This gives a sequence of characters that trigger a terminal operation. For example, a terminal may expect a "magic sequence" to wipe the screen clean, to print in reverse video, or to change the shape of its cursor. Likewise, a terminal may send a "magic sequence" when a particular function key is pressed. For example, the code **klf1=\E5** indicates that this terminal sends the string **<esc>5** when the user presses function-key 1.

Some terminal capabilities may involve *padding* — that is, telling the terminal to delay execution of the capability for a fraction of a second. In some instances, padding may make the difference between a terminal's drawing information correctly, or displaying a jumble.

A delay code can appear anywhere in a string capability code. It is introduced by a dollar sign '$' and enclosed in angle brackets '<>'. The numeric value is always in milliseconds. For example, the code **el=\EK$<3>** indicates that the clear-to-end-of-line code **el** is invoked by the "magic sequence" **<esc>K**, and that it should involve a three-millisecond delay. Function **tputs()** provides the delay.

The delay can be either a number, e.g., "20", or a number followed by an asterisk, e.g., "3*". An asterisk indicates that the padding must be proportional to the number of lines affected by the operation; the amount given is the amount of padding required by each line of output. (This is true even in the case of the insert-character code.) When an asterisk 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.)

The following table gives the commonest **terminfo** capability codes. The *variable* is the name by which the programmer (at the **terminfo** level) accesses the capability. The *code* is the name used in the **terminfo** entry. There is no fixed limit to the length of a code, but the convention is to keep them to five characters or fewer. Whenever possible, names are the same as, or similar to, those in the ANSI Standard X3.64-1979.

The semantics describe features of the code:

† You may specify padding.

†* Padding may be based on the number of lines affected.

The string is passed through **tparm()** with the number of parameters given in the description.

#*i* Indicate the *i*th parameter.

Boolean Codes

Code	*Variable*	*Description*
am	auto_right_margin	Automatic margins
bce	back_color_erase	Erase screen with background color
bw	auto_left_margin	**cub1** wraps from column 0 to last column
ccc	can_change	Terminal can redefine a color

chts	hard_cursor	Cursor is difficult to see
cpix	cpi_changes_res	Changing character pitch also changes resolution
crxm	cr_cancels_micro_mode	Carriage return cancels micro mode
da	memory_above	Display can be retained above the screen
daisy	has_print_wheel	You must change print wheel on printer
db	memory_below	Display can be retained below the screen
eo	erase_overstrike	Erase overstrikes with a blank
eslok	status_line_esc_ok	Escape can be used on the status line
gn	generic_type	Generic line type (e.g., dialup, switch).
hc	hard_copy	Hardcopy terminal
hls	hue_lightness_saturation	Terminal uses HLS color notation
hs	has_status_line	Has an extra "status line"
hz	tilde_glitch	Hazeltine cannot print tildes '~'
in	insert_null_glitch	Insert mode distinguishes NULs
km	has_meta_key	Has a metakey (shift sets parity bit)
mc5i	prtr_silent	Printer does not echo on screen
mir	move_insert_mode	Safe to move while in insert mode
msgr	move_standout_mode	Safe to move in standout modes
npc	no_pad_char	No padding character
nxon	needs_xon_xoff	Padding does not work: needs XON/XOFF
os	over_strike	Terminal overstrikes
sam	semi_auto_right_margin	Printing in last column returns carriage
ul	transparent_underline	Underline character overstrikes
xenl	eat_newline_glitch	Newline ignored after 80 columns (Concept)
xhp	ceol_standout_glitch	Standout not erased by overwriting (HP)
xhpa	col_addr_glitch	Only positive motion for HPA/MHPA capitals
xon	xon_xoff	Terminal uses XON/XOFF handshaking
xsb	no_esc_ctlc	Beehive terminal (F1=escape, F2=**<ctrl-C>**)
xvpa	row_addr_glitch	Only positive motion for VPA/MVPA capitals
xt	teleray_glitch	Tabs destructive, magic SO char (Teleray 1061)

Numeric Codes

Code	*Variable*	*Description*
bufsz	buffer_capacity	Number of bytes buffered before printing
colors	max_colors	Maximum number of colors on the screen
cols	columns	Number of columns in a line
it	init_tabs	Tabs initially every *n* spaces
lines	lines	Number of lines on screen or page
lm	lines_of_memory	Lines of memory if greater than **lines**; zero, variable
maddr	max_micro_address	Maximum value in micro_ ... _address
mjump	max_micro_jump	Maximum value in parm_ ... _micro
mls	micro_line_size	Line-step size when in micro mode
ncv	no_color_video	Video attributes that cannot be used with color
nlab	num_labels	Number of labels on the screen
npins	number_of_pins	Number of pins in the print-head
orc	output_res_char	Horizontal resolution, units per character
orhi	output_res_horz	Horizontal resolution in units per inch
orl	output_res_line	Vertical resolution, units per line
orvi	output_res_vert	Vertical resolution, units per inch
pairs	max_pairs	Maximum number of color_pairs on screen
pb	padding_baud_rate	Lowest baud rate where CR/NL padding is needed
spinh	dot_horz_spacing	Spacing of pins horizontally (pins/inch)
spinv	dot_vert_spacing	Spacing of pins vertically (pins/inch)
vt	virtual_terminal	Virtual terminal number
widcs	wide_char_size	Character step size, double-width mode
wsl	width_status_line	Number of columns in the status line
xmc	magic_cookie_glitch	Number of blank characters left by **smso** or **rmso**

String Capabilities

Code	*Variable*	*Description*

LEXICON

CC	command_character	Terminal-settable command character in prototype
acsc	acs_chars	Pairs of graphical character set (aAbBcC ...)
bel	bell	Audible signal (bell)†
blink	enter_blink_mode	Turn on blinking
bold	enter_bold_mode	Turn on bold (extra bright)
cbt	back_tab	Back tab†
ch	erase_charse	Erase #1 characters†#
chr	change_res_horz	Change horizontal resolution
civis	cursor_invisible	Make cursor invisible
clear	clear_screen	Clear screen†*
cnorm	cursor_normal	Make cursor appear normal (undo **vs** and **vi**)
cpi	change_char_pitch	Change number of characters per inch
cr	carriage_return	Carriage return†*
csnm	char_set_names	Names of character sets
csr	change_scroll_region	change to lines #1 through #2 (vt100)†#
cub	parm_left_cursor	Move cursor left #1 spaces†#
cub1	cursor_left	Move cursor left one space
cud	parm_down_cursor	Move cursor down #1 lines.†*#
cud1	cursor_down	Move cursor down one line
cuf	parm_right_cursor	Move cursor right #1 spaces†*#
cuf1	cursor_right	Move cursor right one space
cup	cursor_address	Cursor motion relative to row 1 column 2†#
cuu	parm_up_cursor	Move cursor up #1 lines†*#
cuu1	cursor_up	Upline (cursor up)
cvr	change_rs_vert	Change vertical resolution
cvvis	cursor_visible	Make cursor very visible
dch	parm_dch	Delete #1 chars†*#
dch1	delete_character	Delete character†*
defc	define_char	Define a character in a character set
dim	enter_dim_mode	Turn on half-bright mode
dl	parm_delete_line	Delete #1 lines†*#
dl1	delete_line	Delete line†*
docr	these_cause_cr	List of characters that trigger carriage return
dsl	dis_status_line	Disable status line
ech	erase_chars	Erase no. 1 characters
ed	clr_eos	Clear to end of display†*
el	clr_eol	Clear to end of line†
el1	clr_bol	Clear to beginning of line, inclusive
enacs	ena_acs	Enable alternate character set
flash	flash_screen	Visible bell (may not move cursor)
ff	form_feed	Hardcopy terminal page eject†*
fsl	from_status_line	Return from status line
hd	down_half_line	Half-line down (forward 1/2 linefeed)
home	cursor_home	Move cursor to home position (if no **cup**)
hpa	column_address	Set cursor column†#
ht	tab	Tab to next eight-space hardware tab stop
hts	set_tab	Set a tab in all rows, current column.
hu	up_half_line	Half-line up (reverse 1/2 linefeed)
ich	parm_ich	Insert #1 blank characters†*#
ich1	insert_character	Insert character†
if	init_file	Name of file containing **is**
il	parm_insert_line	Add #1 new blank lines†*#
il1	insert_line	Add new blank line†*
ind	scroll_forward	Scroll text up†
indn	parm_index	Scroll forward #1 lines†#
initc	initialize_color	Initialize color definition
initp	initialize_pair	Initialize color pairs
invis	enter_secure_mode	Turn on blank mode (characters invisible)
ip	insert_padding	Insert pad after character inserted†*
iprog	init_prog	Full path name of initialization program
is1	init_1string	Terminal-initialization string
is2	init_2string	Terminal-initialization string

is3	init_3string	Terminal-initialization string
kBEG	key_sbeg	Sent by shifted beginning key
kCAN	key_scancel	Sent by shifted cancel key
kCMD	key_scommand	Sent by shifted command key
kCPY	key_scopy	Sent by shifted copy key
kCRT	key_screate	Sent by shifted create key
kDC	key_sdc	Sent by shifted delete-character key
kDL	key_sdl	Sent by shifted delete-line key
kEND	key_send	Sent by shifted end key
kEOL	key_seol	Sent by shifted EOL clear-line key
kEXT	key_sexit	Sent by shifted exit key
kFND	key_sfind	Sent by shifted find key
kHLP	key_shelp	Sent by shifted help key
kHOM	key_shome	Sent by shifted home key
kIC	key_sic	Sent by shifted input key
kLFT	key_sleft	Sent by shifted (æ) key
kMOV	key_smove	Sent by shifted move key
kMSG	key_smessage	Sent by shifted message key
kNXT	key_snext	Sent by shifted next key
kOPT	key_soptions	Sent by shifted option key
kPRT	key_sprint	Sent by shifted print key
kPRV	key_sprevious	Sent by shifted previous key
kRDO	key_sredo	Sent by shifted redo key
kRES	key_srsume	Sent by shifted resume key
kRIT	key_sright	Sent by shifted (Æ) key
kRPL	key_sreplace	Sent by shifted replace key
kSAV	key_ssave	Sent by shifted save key
kSPD	key_ssuspend	Sent by shifted suspend key
kUND	key_sundo	Sent by shifted undo key
ka1	key_a1	Sent by key A1, upper left of keypad
ka3	key_a3	Sent by key A3, upper right of keypad
kb2	key_b2	Sent by key B2, center of keypad
kbeg	key_beg	Sent by "begin" key
kbs	key_backspace	Sent by backspace key
kc1	key_c1	Sent by key C1, lower left of keypad
kc3	key_c3	Sent by key C3, lower right of keypad
kcan	key_cancel	Sent by cancel key
kcbt	key_btab	Sent by back-tab key
kclo	key_close	Sent by close key
kclr	key_clear	Sent by clear-screen or erase key
kcmd	key_command	Sent by "cmd" key
kcpy	key_copy	Sent by copy key
kcrt	key_create	Sent by create key
kctab	key_ctab	Sent by clear-tab key
kcub1	key_left	Sent by (æ) key
kcud1	key_down	Sent by (°) key
kcuf1	key_right	Sent by (Æ) key
kcuu1	key_up	Sent by terminal (ª) key
kdch1	key_dc	Sent by delete-character key
kdl1	key_dl	Sent by delete-line key
ked	key_eos	Sent by clear-to-end-of-screen key
kel	key_eol	Sent by clear-to-end-of-line key
kend	key_end	Sent by end key
kent	key_enter	Sent by (¢) key
kext	key_exit	Sent by exit key
kf0	key_f0	Sent by function key 0
kf1	key_f1	Sent by function key 1
kf10	key_f10	Sent by function key 10
kf11	key_f11	Sent by function key 11
kf12	key_f12	Sent by function key 12
kf13	key_f13	Sent by function key 13
kf14	key_f14	Sent by function key 14

kf15	key_f15	Sent by function key 15
kf16	key_f16	Sent by function key 16
kf17	key_f17	Sent by function key 17
kf18	key_f18	Sent by function key 18
kf19	key_f19	Sent by function key 19
kf2	key_f2	Sent by function key 2
kf20	key_f20	Sent by function key 20
kf21	key_f21	Sent by function key 21
kf22	key_f22	Sent by function key 22
kf23	key_f23	Sent by function key 23
kf24	key_f24	Sent by function key 24
kf25	key_f25	Sent by function key 25
kf26	key_f26	Sent by function key 26
kf27	key_f27	Sent by function key 27
kf28	key_f28	Sent by function key 28
kf29	key_f29	Sent by function key 29
kf3	key_f3	Sent by function key 3
kf30	key_f30	Sent by function key 30
kf31	key_f31	Sent by function key 31
kf32	key_f32	Sent by function key 32
kf33	key_f33	Sent by function key 33
kf34	key_f34	Sent by function key 34
kf35	key_f35	Sent by function key 35
kf36	key_f36	Sent by function key 36
kf37	key_f37	Sent by function key 37
kf38	key_f38	Sent by function key 38
kf39	key_f39	Sent by function key 39
kf4	key_f4	Sent by function key 4
kf40	key_f40	Sent by function key 40
kf41	key_f41	Sent by function key 41
kf42	key_f42	Sent by function key 42
kf43	key_f43	Sent by function key 43
kf44	key_f44	Sent by function key 44
kf45	key_f45	Sent by function key 45
kf46	key_f46	Sent by function key 46
kf47	key_f47	Sent by function key 47
kf48	key_f48	Sent by function key 48
kf49	key_f49	Sent by function key 49
kf5	key_f5	Sent by function key 5
kf50	key_f50	Sent by function key 50
kf51	key_f51	Sent by function key 51
kf52	key_f52	Sent by function key 52
kf53	key_f53	Sent by function key 53
kf54	key_f54	Sent by function key 54
kf55	key_f55	Sent by function key 55
kf56	key_f56	Sent by function key 56
kf57	key_f57	Sent by function key 57
kf58	key_f58	Sent by function key 58
kf59	key_f59	Sent by function key 59
kf6	key_f6	Sent by function key 6
kf60	key_f60	Sent by function key 60
kf61	key_f61	Sent by function key 61
kf62	key_f62	Sent by function key 62
kf63	key_f63	Sent by function key 63
kf7	key_f7	Sent by function key 7
kf8	key_f8	Sent by function key 8
kf9	key_f9	Sent by function key 9
kfnd	key_find	Sent by find key
khlp	key_help	Sent by help key
khome	key_home	Sent by home key
khts	key_stab	Sent by set-tab key
kich1	key_ic	Sent by insert char/enter insert-mode key

kil1	key_il	Sent by insert line
kind	key_sf	Sent by scroll-forward/down key
kll	key_ll	Sent by home-down key
kmrk	key_mark	Sent by mark key
kmsg	key_message	Sent by message key
kmov	key_move	Sent by move key
knp	key_npage	Sent by next-page key
knxt	key_next	Sent by next-object key
kopn	key_open	Sent by open key
kopt	key_options	Sent by options key
kpp	key_ppage	Sent by previous-page key
kprt	key_print	Sent by print (copy) key
kprv	key_previous	Sent by previous-object key
krdo	key_redo	Sent by redo key
kref	key_reference	Sent by reference key
kres	key_resume	Sent by resume key
krfr	key_refresh	Sent by refresh key
kri	key_sr	Sent by scroll-backward/up key
krmir	key_eic	Sent by **rmir** or **smir** in insert mode
krpl	key_replace	Sent by replace key
krst	key_restart	Sent by restart key
ksav	key_save	Sent by save key
kslt	key_select	Sent by select key
kspd	key_suspend	Sent by suspend key
ktbc	key_catab	Sent by clear-all-tabs key
kund	key_undo	Sent by undo key
lf0	label_f0	Label on function key 0 if not F0
lf1	label_f1	Label on function key 1 if not F1
lf10	label_f10	Label on function key 10 if not F10
lf2	label_f2	Label on function key 2 if not F2
lf3	label_f3	Label on function key 3 if not F3
lf4	label_f4	Label on function key 4 if not F4
lf5	label_f5	Label on function key 5 if not F5
lf6	label_f6	Label on function key 6 if not F6
lf7	label_f7	Label on function key 7 if not F7
lf8	label_f8	Label on function key 8 if not F8
lf9	label_f9	Label on function key 9 if not F9
ll	cursor_to_ll	Last line, first column (if no **cup**)
lpi	change_line_pitch	Change number of lines per inch
mc0	print_screen	Print contents of the screen
mc4	prtr_off	Turn off printer
mc5	prtr_on	Turn on printer
mcub	parm_left_micro	Like **cub** for micro adjustment
mcub1	micro_left	Like **cub1** for micro adjustment
mcud	parm_down_micro	Like **cud** for micro adjustment
mcud1	micro_down	Like **cud1** for micro adjustment
mcuf	parm_right_micro	Like **cuf** for micro adjustment
mcuf1	micro_right	Like **cuf1** for micro adjustment
mcuu	parm_up_micro	Like **cuu** for micro adjustment
mcuu1	micro_up	Like **cuu1** for micro adjustment
mgc	clear_margins	Clear all margins (top, bottom, sides)
mhpa	micro_column_address	Like **hpa** for micro adjustment
mrcup	cursor_mem_address	Memory-relative cursor addressing
mvpa	micro_row_address	Like **vpa** for micro adjustment
nel	newline	Newline (behaves like CR followed by LF)
oc	orig_colors	Set all colors to originals
op	orig_pair	Set default color_pair to original
pad	pad_char	Pad character (rather than NUL)
pfkey	pkey_key	Program function key 1 to type string 2
pfloc	pkey_local	Program function key 1 to execute string 2
pfx	pkey_xmit	Program function key 1 to transmit string 2
pln	plab_norm	Program label 1 to show string 2

porder	order_of_pins	Match software bits to print-head pins
prot	enter_protected_mode	Turn on protected mode
rc	restore_cursor	Restore cursor to position of last **sc**
rep	repeat_char	Repeat character #1 #2 times. †*#
rev	enter_reverse_mode	Turn on reverse-video
rf	reset_file	Name of file containing reset string
rfi	reg_for_input	Send next input character
ri	scroll_reverse	Scroll text down†
rin	parm_rindex	Scroll backward one line†#
ritm	exit_italics_mode	Disable italics
rlm	exit_leftward_mode	Enable rightward motion
rmacs	exit_alt_charset_mode	End alternate character set†
rmam	exit_am_mode	Turn off automatic margins
rmcup	exit_ca_mode	String to end programs that use **cup**
rmdc	exit_delete_mode	End delete mode
rmicm	exit_micro_mode	Disable micro-motion capabilities
rmir	exit_insert_mode	End insert mode
rmkx	keypad_local	Exit “keypad transmit” mode
rmln	label_off	Turn off soft labels
rmm	meta_off	Turn off “meta mode”
rmp	char_padding	Like **ip**, but in replace mode
rmso	exit_standout_mode	End stand out mode
rmul	exit_underline_mode	End underscore mode
rmxon	exit_xon_mode	Turn off XON/XOFF handshaking
rs1	reset_1string	Reset terminal completely to sane modes
rs2	reset_2string	Reset terminal completely to sane modes
rs3	reset_3string	Reset terminal completely to sane modes
rshm	exit_shadow_mode	Disable shadow printing
rsubm	exit_subscript_mode	Disable subscript printing
rsupm	exit_superscript_mode	Disable superscript printing
rum	exit_upward_motion	Enable downward motion
rwidm	exit_doublewide_mode	Disable double-width printing
sbim	start_bit_margin	Start printing bit-mapped graphics
sc	save_cursor	Save cursor position†
scp	set_color_pair	Set current color pair
scs	select_char_set	Select character set
scsd	start_char_set_def	Start definition of a character set
sdrfq	enter_draft_quality	Set draft-quality printing
setb	set_background	Set current background color
setf	set_foreground	Set current foreground color
sgr	set_attributes	Define the nine video attributes†*#
sgr0	exit_attribute_mode	Turn off all attributes
sitm	enter_italics_mode	Enable italics
slm	enter_leftward_mode	Enable leftward carriage motion
smacs	enter_alt_charset_mode	Start alternate character set†
smam	enter_am_mode	Turn on automatic margins
smcup	enter_ca_mode	String to begin programs that use **cup**
smdc	enter_delete_mode	Delete mode (enter)
smgb	set_bottom_margin	Set bottom margin to current line
smgbp	set_bottom_margin_parm	Set bottom margin at lines 1 or 2
smgl	set_left_margin	Set left margin to current column
smglp	set_left_margin_parm	Set left margin to columns 1 or 2
smgr	set_right_margin	Set right margin to current column
smgrp	set_right_margin_parm	Set right margin to columns 1 or 2
smgt	set_top_margin	Set top margin to current line
smgtp	set_top_margin_parm	Set top margin to lines 1 or 2
smicm	enter_micro_mode	Enable micro-motion capabilities
smir	enter_insert_mode	Insert mode (enter)
smkx	keypad_xmit	Enter “keypad transmit” mode
smln	label_on	Turn on soft labels
smm	meta_on	Turn on “meta mode” (eighth bit)
smso	enter_standout_mode	Begin stand-out mode

smul	enter_underline_mode	Start underscore mode
smxon	enter_xon_mode	Turn on XON/XOFF handshaking
snlq	enter_near_letter_quality	Set near-letter-quality printing
snrmq	enter_normal_quality	Set normal-quality print
sshm	enter_shadow_mode	Enable shadow printing
ssubm	enter_subscript_mode	Enable subscript printing
ssupm	enter_superscript_mode	Enable superscript printing
subcs	subscript_characters	List of "subscript-able" characters
sum	enter_upward_mode	Enable upward carriage motion
supcs	superscript_characters	List of "superscript-able" characters
swidm	enter_doublewide_mode	Enable double-wide printing
tbc	clear_all_tabs	Clear all tab stops†
tsl	to_status_line	Go to status line, column 1
uc	underline_char	Underscore one char and move past it
vpa	row_address	Vertical position absolute (set row)†#
wind	set_window	Current window is lines #1-#2, columns 3—4
xoffc	xoff_character	XOFF character
xonc	xon_character	XON character
zerom	zero_motion	No motion for subsequent character

Escape Sequences

You can use the following escape sequences with any string-capability entry:

\E	**<esc>** character
\e	**<esc>** character
^*X*	**<ctrl-***X***>** for any appropriate *X*
\n	Newline
\r	Carriage return
\t	Horizontal tab
\b	Backspace
\f	Formfeed
\s	Space
*****000*	Value of a character in three octal digits
\^	Literal carat
\,	Literal comma
****	Literal backslash

Parameterized Strings

Cursor-addressing and other strings requiring parameters in the terminal are described by a parameterized string capability, with **printf()**-like escape sequences in it. Each escape sequence is introduced with a percent sign '%', followed by one character that described the type of formatting to be performed, as follows:

%%	Literal '%'
%d	Decimal integer
%2d	Decimal integer with at least two places
%02d	Decimal integer, two places, zero padding
%3d	Decimal integer with at least three places
%03d	Decimal integer, three places, zero padding
%c	Character
%s	String
%p[*i***]**	Push *i*th parameter
%P[*a-z***]**	Set variable [*a-z*] to **pop()**
%g[*a-z***]**	Push variable [*a-z*]
%'*c***'**	Character constant *c*
%{*nn***}**	Integer constant *nn*

%+	Addition: **push(pop() + pop())**
%-	Subtraction: **push(pop() - pop())**
%*	Multiplication: **push(pop() * pop())**
%/	Division: **push(pop() / pop())**
%m	Modulo: **push(pop() % pop())**
%&	Bitwise AND: **push(pop() op pop())**
%|	Bitwise OR: **push(pop() op pop())**
%^	Bitwise NOR: **push(pop() op pop())**
%=	Logical AND: **push(pop() op pop())**
%>	Logical OR: **push(pop() op pop())**
%<	Logical NOR: **push(pop() op pop())**
%!	Unary NOT: **push(op pop())**
%~	Unary complement: **push(op pop())**
%i	Add one to first two parmameters (for ANSI terminals)

The parameterized mechanism is based on a stack. **%** operations push parameters and constants onto the stack, do arithmetic and other operations on the top of the stack, and print out values in various formats. Up to nine parameters can be used at once. If-then-else testing is possible, as is storage in a limited number of variables. There is no provision for loops or printing strings in any format other than **%s**.

For example, compare the **termcap** entry **cm** and the **terminfo** entry **cup**. **%+** (add space and print as a character) **cm** would be treated as **%p1%' '%+%c**, that is, push the first parameter, push space, add the top two numbers onto the stack, and output the top item on the stack using character (**%c**) format. For the second parameter, change **%p1** to **%p2**. **%.** (print as a character) becomes **%p1%c**. **%d** (print in decimal) becomes **%p1%d**.

As with **tgoto()**, characters standing by themselves (no '%' sign) are output as is.

Alternate Characters

The instruction **acsc** defines a set of alternate characters. These alternate characters define, among other things, the characters used to draw boxes.

acsc is set to a string composed of pairs of characters. The first character in each pair gives the character used by a VT100 in graphics mode to display; the second character is the one for the terminal in use. The following table shows the VT100 graphic-character set:

Arrow right	**+**
Arrow left	**,**
Arrow down	**.**
Full block (inverted space)	**0**
Lantern	**I**
Arrow up	**-**
Diamond	**'**
Checkboard	**a**
Degree	**f**
+/- Sign	**g**
Centered rectangles	**h**
Lower right corner	**j**
Upper right corner	**k**
Upper left corner	**l**
Lower left corner	**m**
Cross	**n**
Upper horizontal line	**o**
Middle horizontal line	**q**
Lower horizontal line	**s**
Left tee	**t**
Right tee	**u**
Lower tee	**v**
Upper tee	**w**
Vertical line	**x**
Centered dot	**~**

Changes from termcap to terminfo

This section describes features of **terminfo** that **termcap** does not contain.

Defaults

terminfo does not contain every default found in **termcap**. **termcap,** for example, assumed that \r was a carriage return unless **nc** was present, indicating that it did not work, or **cr** was present, indicating an alternative. In **terminfo**, if **cr** is present, the string so given works; otherwise it should be assumed *not* to work. The **bs** and **bc** capabilities are replaced by **cub** and **cub1**. (The former takes a parameter, moving left that many spaces. The latter is probably more common in terminals and moves left one space.) **nl** (linefeed) has been split into two functions: **cud1** (moves the cursor down one line) and **ind** (scroll forward). **cud1** applies when the cursor is not on the bottom line, **ind** applies when it is on the bottom line. The bell capability is now explicitly given as **bel**.

The **terminfo** data base is compiled, unlike termcap. This means that a **terminfo** source file (describing some set of terminals) is processed by the terminfo compiler, producing a binary description of the terminal in a file under **/usr/lib/terminfo**. The function **setupterm()** reads this file. The advantage to compilation is that starting up a program using **terminfo** is faster. The increase in speed comes partly from not having to skip past other terminal descriptions, and partly from the compiler having sorted the capabilities into order so that a linear scan can read them in.

The **terminfo** compiler **tic** uses the environment variable **TERMINFO** to be the destination directory of the new object files. It is also used by **setupterm()** to find an entry for a given terminal. First it looks in the directory given in **TERMINFO** and, if not found there, checks **/usr/lib/terminfo**.

Basic Example

The following gives the **terminfo** description for a simple terminal, the Lear-Siegler ADM-3:

```
adm3|3|lsi adm3,
      cr=^M, cud1=^J, ind=^J, bel=^G,
      am, cub1=^H, clear=^Z, lines#24, cols#80
```

As you can see, the description is divided into comma-separated fields. The following discusses each field in detail.

adm3 | 3 | lsi adm3,
: The first field names the terminal. This field is unique in that it is divided into a number of sub-fields, which are separated by vertical bar characters. The first sub-field gives the name by which the terminal is normally addressed in a program; the last gives a longer, descriptive name.

cr=^M, To move the cursor to the left margin, send **<ctrl-M>**.

cud1=^J, To move the cursor down one row, send **<ctrl-J>**.

ind=^J, To scroll the screen up, send **<ctrl-J>**. Note that the ADM-3, like most terminals, does not scroll unless the cursor is on the last row.

bel=^G, To ring the terminal's bell, send **<ctrl-G>**.

am, This boolean code indicates that the ADM-3 wraps to the leftmost column of the of the next row when the cursor reaches the rightmost column.

cub1=^H, To move the cursor nondestructively one column to the left, send **<ctrl-H>**.

clear=^Z, To clear the screen, send **<ctrl-Z>**.

lines#24, The ADM-3 has 24 rows (lines).

cols#80, The ADM-3 has 80 columns.

Modifying an Entry

A full discussion of how to modify a **terminfo** entry is beyond the scope of this article. The references, below, name several volumes that discuss this topic at length.

In brief, modifying a **terminfo** entry requires that you use the command **infocmp** to de-compile the entry for a given terminal, modify the text by hand, then use the command **tic** to recompile and re-install the entry.

C-Level Routines

The library **/usr/lib/libcurses.a** contains a suite of C functions with which you can read a given terminal's **terminfo** capabilities. You must reference the **terminfo** capabilities in your program as global variables whose names are identical to the full names of the capabilities themselves; e.g., **auto_left_margin**. These functions are

declared in the header files **<curses.h> <term.h>,** and **<terminfo.h>**; note that you must **#include** all three header files in your C program.

You can call the following functions from within a C program to read a **terminfo** entry:

fixterm()	Set the terminal into program mode
putp()	Write a string into *stdwin*
resetterm()	Reset the terminal into a saved mode
setupterm()	Initialize terminal capabilities
tparm()	Output a parameterized string
tputs()	Process a capability string
vidattr()	Set the terminal's video attributes
vidputs()	Set video attributes into a function

setupterm() initializes a terminal. It inhales all terminal capabilities at once, and performs all other system-dependent initialization.

A program should call **resetterm()** when it exits or calls a shell escape, to restore the tty modes. When it returns from a shell escape, the program should call **fixterm()** to set the tty modes back to their internal settings.

tparm() is a more powerful, parameterized string mechanism. It resembles the **termcap** function **tgoto()**. **tgoto()** is still available for compatibility. **tputs()** is unchanged.

Files

/usr/lib/libcurses.a — Routines for reading **terminfo** descriptions
/usr/lib/terminfo/?/* — Directories containing compiled descriptions

See Also

Administering COHERENT, captoinfo, curses, fixterm(), infocmp, putp(), resetterm(), setupterm(), term, termcap, tic, tparm(), tputs(), vi, vidputs()
Strang, J., Mui, L., O'Reilly, T.: *Termcap and Terminfo*. Sebastopol, CA: O'Reilly & Associates, Inc., 1991. *Highly recommended.*

Notes

As mentioned above, each **terminfo** description is kept in its own file, in a subdirectory of directory **/usr/lib/terminfo**. Each file is named after the device it describes. Thus, to see what terminal devices have **terminfo** descriptions, type the command:

```
ls -laR /usr/lib/terminfo
```

You may wish to redirect the output of this command into a file, for further study later on.

This implementation of **terminfo** was written by Pavel Curtis of Cornell University. It was ported to COHERENT by Udo Munk.

termio — Device Driver

General terminal interface

COHERENT uses two methods for controlling terminals: **sgtty** and **termio**. To use **sgtty**, simply include the statement **#include <sgtty.h>** in your sources. To use **termio**, include the statement **#include <termio.h>**.

The rest of this article discusses the **termio** method of controlling terminals.

When a terminal file is opened, it normally causes the process to wait until a connection is established. In practice, users' programs seldom open these files: they are opened by the program **getty** and become a user's standard input, output, and error files. The very first terminal file opened by the process group leader of a terminal file not already associated with a process group becomes the *control terminal* for that process group. The control terminal plays a special role in handling **quit** and **interrupt** signals, as discussed below. The control terminal is inherited by a child process during a call to **fork()**. A process can break this association by changing its process group using **setpgrp()**.

A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters can be typed at any time, even while output is occurring, and are lost only when the system's input buffers become completely full, which is rare, or when the user has accumulated the maximum allowed number of input characters that have not yet been read by some program. Currently, this limit is 256 characters. When the input limit is reached, the system throws away all the saved characters without notice.

Normally, terminal input is processed in units of lines. A line is delimited by a newline character (ASCII LF), an end-of-file character (ASCII EOT), or an end-of-line character. This means that a program attempting to read will be suspended until an entire line has been typed. Also, no matter how many characters are requested in the read call, at most one line is returned. It is not, however, necessary to read a whole line at once; any number of characters may be requested in a read, even one, without losing information.

During input, the system normally processes **erase** and **kill** characters. By default, the backspace character erases the last character typed, except that it will not erase beyond the beginning of the line. By default, the **<ctrl-U>** kills (deletes) the entire input line, and optionally outputs a newline character. Both these characters operate on a keystroke-by-keystroke basis, independently of any backspacing or tabbing which may have been done. Both the erase and kill characters may be entered literally by preceding them with the escape character (\). You can change the erase and kill characters.

Certain characters have special functions on input. These functions and their default character values are summarized as follows:

INTR (**<ctrl-C>** or ASCII ETX) generates an *interrupt* signal that is sent to all processes with the associated control terminal. Normally, each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location. For details and a table of legal signals, see the Lexicon entry for **signal()**.

QUIT (**<ctrl-\>** or ASCII ES) generates a *quit* signal. Its treatment is identical to that of the interrupt signal except that, unless a receiving process has made other arrangements, it not only terminates but dumps a core image file (named **core**) into the current working directory.

ERASE (**<backspace>** or ASCII BS) erases the preceding character. It does not erase beyond the start of a line, as delimited by a newline, **EOF**, or EOL character.

KILL (**<ctrl-U>** or ASCII NAK) deletes the entire line, as delimited by a newline, **EOF**, or EOL character.

EOF (**<ctrl-D>** or ASCII EOT) generates an end-of-file character from a terminal. When received, all the characters waiting to be read are immediately passed to the program without waiting for a newline, and the **EOF** is discarded. Thus, if no characters are waiting, which is to say the **EOF** occurred at the beginning of a line, zero characters are passed back; this is the standard end-of-file indication.

NL (ASCII LF) is the normal line delimiter. It cannot be changed or escaped.

EOL (ASCII LF) is an additional line delimiter, like NL. It is not normally used.

STOP (**<ctrl-S>** or ASCII DC3) can be used to suspend output. It is useful with CRT terminals to prevent output from disappearing before it can be read. While output is suspended, **STOP** characters are ignored and not read.

START (**<ctrl-Q>** or ASCII DC1) resumes output that has been suspended by a **STOP** character. While output is not suspended, **START** characters are ignored and not read. The **START/STOP** characters cannot be changed or escaped.

You can change the character values for **INTR**, **QUIT**, **ERASE**, **KILL**, **EOF**, and **EOL** To suit your taste. The **ERASE**, **KILL**, and **EOF** character can be preceded by a '\' character, in which case the system ignores its special meaning.

When the carrier signal from the data-set drops, the system sends a *hangup* signal to all processes that have this terminal as their control terminal. Unless other arrangements have been made, this signal causes the process to terminate. If the hangup signal is ignored, any subsequent read returns EOF. Thus, programs that read a terminal and test for end-of-file can terminate appropriately when hung up on.

When one or more characters are written, they are transmitted to the terminal as soon as previously written characters have finished typing. Input characters are echoed by putting them into the output queue as they arrive. If a process produces characters more rapidly than they can be printed, it is suspended when its output queue exceeds a preset limit. When the queue has drained down to that threshold, the program resumes.

Several calls to **ioctl()** apply to terminal files. The primary calls use the following structure, defined in **<termio.h>**:

```
#define NCC 8
struct termio {
        unsigned short c_iflag;    /* input modes */
        unsigned short c_oflag;    /* output modes */
        unsigned short c_cflag;    /* control modes */
        unsigned short c_lflag;    /* local modes */
        char           c_line;     /* line discipline */
        unsigned char  c_cc[NCC];  /* control chars */
};
```

The special control characters are defined by the array **c_cc**. The relative positions and initial values for each function are as follows:

```
       0   INTR      ^C
       1   QUIT      ^\
       2   ERASE     \b
       3   KILL      ^U
       4   EOF       ^D
       5   EOL       \n
       6   reserved
       7   reserved
```

The field **c_iflag** describes the basic terminal input control:

BRKINT	Signal interrupt on break
IGNPAR	Ignore characters with parity errors
INPCK	Enable input parity check
ISTRIP	Strip character
ICRNL	Map CR to NL on input
IXON	Enable start/stop output control
IXOFF	Enable start/stop input control

If **INPCK** is set, input parity checking is enabled. If it is not set, then checking is disabled. This allows output parity generation without input parity errors.

If **ISTRIP** is set, valid input characters are stripped to seven bits before being processed; otherwise, all eight bits are processed.

If **IXON** is set, **START/STOP** output control is enabled. A received **STOP** character suspends output and a received **START** character restarts output. All start/stop characters are ignored and not read.

If **IXOFF** is set, the system transmits **START/STOP** characters when the input queue is nearly empty or nearly full.

The initial input control value is all bits clear.

The field **c_oflag** field specifies the system treatment of output:

OPOST	Postprocess output.
OLCUC	Map lower case to upper on output.
ONLCR	Map NL to CR-NL on output.

If **OPOST** is set, output characters are post-processed as indicated by the remaining flags; otherwise, characters are transmitted without change.

If **OLCUC** is set, a lower-case alphabetic character is transmitted as the corresponding upper-case character. This function is often used with **IUCLC**.

If **ONLCR** is set, the NL character is transmitted as the CR-NL character pair.

The initial output control value is all bits clear.

The field **c_cflag** describes the hardware control of the terminal, as follows:

CBAUD	Baud rate
B0	Hang up
B50	50 baud
B75	75 baud
B110	110 baud
B134	134.5 baud

B150	150 baud
B200	200 baud
B300	300 baud
B600	600 baud
B1200	1200 baud
B1800	1800 baud
B2400	2400 baud
B4800	4800 baud
B9600	9600 baud
B19200	19200 baud
B38400	38400 baud
CREAD	Enable receiver
PARENB	Parity enable
PARODD	Odd parity, else even
HUPCL	Hang up on last close
CLOCAL	Local line, else dial-up

The **CBAUD** bits specify the baud rate. The zero-baud rate, **B0**, hangs up the connection. If **B0** is specified, the data-terminal-ready signal is not asserted. Normally, this disconnects the line. For any particular hardware, the system ignores impossible changes to the speed.

If **PARENB** is set, parity generation and detection is enabled and a parity bit is added to each character. If parity is enabled, the **PARODD** flag specifies odd parity if set; otherwise, even parity is used.

If **CREAD** is set, the receiver is enabled. Otherwise, no characters will be received.

If **HUPCL** is set, COHERENT disconnects the line when the last process with the line open closes the line or terminates; that is, the data-terminal-ready signal is not asserted.

If **CLOCAL** is set, the system assumes that the line to be a local, direct connection with no modem control. Otherwise, it assumes modem control.

The line discipline uses the field **c_lflag** to control terminal functions. The basic line discipline (zero) provides the following:

ISIG	Enable signals
ICANON	Canonical input (erase and kill processing)
XCASE	Canonical upper/lower presentation
ECHO	Enable echo
ECHOE	Echo erase character as BS-SP-BS
ECHOK	Echo NL after kill character
ECHONL	Echo NL

The following gives the meaning of each flag in detail:

ISIG If this flag is set, the system checks each input character against the special control characters **INTR** and **QUIT**. If an input character matches one of these control characters, the system executes the function associated with that character. If it is not set is not set, the system performs no checking; thus, these special input functions are possible only if **ISIG** is set. You can disable these functions individually by changing the value of the control character to an unlikely or impossible value (e.g., 0377).

ICANON

If this flag is set, the system enables canonical processing. This enables the erase and kill-edit functions, and limits the assembly of input characters into lines delimited by NL, EOF, and EOL. The system also interprets the *vmin* and *vtime* locations in the **termio** structure as **c_cc[VEOF]** and **c_cc[VEOL]**, respectively.

When the **ICANON** bit is cleared, you must set **c_cc[VMIN]** and **c_cc[VTIME]** to appropriate *vmin* and *vtime* values. *vmin* is a number from 0 to 255 that gives the minimum number of characters required before any read operation completes. *vtime* is a number from 0 to 255 that specifies how long, in tenths of a second, to wait for completion of input. The following describes how **termio** processes the *vmin* and *vtime* values:

1. If *vmin* is greater than zero and *vtime* equals zero, block until *vmin* characters are received.
2. If both *vmin* and *vtime* are greater than zero, block until the first character is received, then return after *vmin* characters are received or *vtime*/10 seconds have elapsed since the last character was received, whichever occurs first.

3. If *vmin* equals zero, return after first character is received or after *vtime*/10 seconds have passed, whichever occurs first. It may return a read count of zero — but will return one character if it is available, even if *vtime* is zero.

You can use the command **stty** to reset the *vmin* and *vtime* values. The header file **termio.h** includes the constants **VMIN** and **VTIME**, which set default values for *vmin* and *vtime*, respectively.

XCASE If this flag is set, and if **ICANON** is set, an upper-case letter is accepted on input by preceding it with a '\' character, and is output preceded by a '\' character. In this mode, the following escape sequences are generated on output and accepted on input:

For:	*Use:*
`	\'
|	\!
~	\^
{	\(
}	\)
\	\\

For example, **A** is input as **\a**, **\n** as **\\n**, and **\N** as **\\\n**.

ECHO If this flag is set, characters are echoed as received. When **ICANON** is set, the following echo functions are possible:

- If **ECHO** and **ECHOE** are set, the erase character is echoed as ASCII BS SP BS, which clears the last character from the screen.
- If **ECHOE** is set and **ECHO** is not set, the erase character is echoed as ASCII SP BS.
- If **ECHOK** is set, the NL character is echoed after the kill character to emphasize that the line will be deleted. Note that an escape character preceding the erase or kill character removes any special function.
- If **ECHONL** is set, the NL character is echoed even if **ECHO** is not set. This is useful for terminals set to local echo ("half duplex").

Unless escaped, the EOF character is not echoed. Because EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.

The initial line-discipline control value is all bits clear.

The primary calls to **ioctl()** have the following form:

```
ioctl( fildes, command, arg )
struct termio *arg;
```

The following commands use this form:

TCGETA Get the parameters associated with the terminal and store in the **termio** structure referenced by **arg**.

TCSETA Set the parameters associated with the terminal from the structure referenced by **arg**. The change is immediate.

TCSETAW Wait for the output to drain before setting the new parameters. This form should be used when changing parameters that affect output.

TCSETAF Wait for the output to drain, then flush the input queue and set the new parameters.

Additional calls to **ioctl()** have the following form:

```
ioctl( fildes, command, arg )
int arg;
```

The following command uses this form:

TCFLSH
Flush both the input and output queues.

Note that header **<termio.h>** defines other constants for purposes of portability. Features designated by these constants are unavailable in the current release of COHERENT 386.

Example

The following example gives some functions that let you perform a non-blocking read of the keyboard — that is, a

```
read(0, &c, sizeof(char));
```

that returns zero (failure) rather than waiting for input if there is no current typed character.

To do so, you must do the following:

- Set up keyboard input appropriately with the ioctls **TCGETA** and **TCSETA**.
- Turn off **ICANON**.
- Turn off the various versions of **ECHO**.
- Use **ISIG** to disable keyboard interrupts.
- Finally, set:

```
termiob.c_cc[VMIN] = 0;
termiob.c_cc[VTIME] = 0;
```

This lets **read()** return after reading zero bytes in .0 seconds.

```
#include <termio.h>
#include <stdlib.h>

void
ttyinit()
{
      struct      termio      termiob;

      ioctl(0, TCGETA, &termiob); /* get tty characteristics */
      termiob.c_cc[VMIN] = 0;
      termiob.c_cc[VTIME] = 0; /* non-blocking read */
      ioctl(0, TCSETA, &termiob); /* set new mode */
}

int
ttycheck()
{
      static int done = 0;
      char c;

      if (done)
            return 0;
      if (read(0, &c, 1) != 0) {
            if (c == 'a')
                  return 0;
            else if (c != ' ') {
                  ++done;
                  return 0;
            }

            /* After <space>, pause until another character is typed */
            while (read(0, &c, 1) == 0)
                  ;
      }
      return 1;
}

main()
{
      ttyinit();

      while (1) {
            printf("Still checking ...\n");
            if (!ttycheck())
                  exit(EXIT_SUCCESS);
      }
}
```

LEXICON

For another example of how to manipulate the **termio** structure, see the entry for **ioctl()**.

Files

/dev/tty*

See Also

device drivers, ioctl(), stty, terminal, termio.h, termios
POSIX Standard, §7.1

Notes

The version of **stty** that is supplied with COHERENT 386 provides complete access to the System-V-style **termio** structure. This lets you specify and view any combination of the fields therein, including various delays. How these fields are processed, however, depends on the device in question. The settings of **termio** are processed by the kernel's in-line discipline and device-driver modules. In COHERENT 4.0.1, none of these modules pays attention to delay settings.

termio.h — Header File

Definitions used with terminal input and output
#include <termio.h>

termio.h defines structures and constants used by functions that control terminal input and output.

See Also

header files, termio
POSIX Standard, §7.1.2

Notes

COHERENT lets you choose between **sgtty** and **termio** to control terminals. For more information, see the Lexicon entries for **sgtty** and **termio**.

termios — Overview

POSIX extended terminal interface

The name **termios** describes a group of routines that POSIX Standard defines to extend the **termio** interface to terminals. **termios** includes the following routines:

cfgetispeed() . . . . . Get input speed
cfgetospeed() . . . . . Get output speed
cfsetispeed(). . . . . Set input speed
cfsetospeed() . . . . . Set output speed
tcdrain() . . . . . Drain output to a device
tcflow() . . . . . Control flow on a terminal device
tcflush() . . . . . Flush data being exchanged with a terminal
tcgetattr() . . . . . Get terminal attributes
tcsendbreak() . . . . . Send a break to a terminal
tcsetattr() . . . . . Set terminal attributes

Each is described in its own Lexicon entry. Under COHERENT, all are defined as macros in header file **<termios.h>**.

Example

The following example returns the input and output speeds for the terminal device that you now are using:

```
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <termios.h>

int main()
{
      struct termios term;
      int speed;
```

```
	if (tcgetattr(STDIN_FILENO, &term) < 0) {
		fprintf(stderr, "tcgetattr error");
		exit(EXIT_FAILURE);
	}

	speed = cfgetispeed(&term);
	printf("tty line input speed is ");

	if	(speed == B50)		printf("50 baud\n");
	else if (speed == B75)		printf("75 baud\n");
	else if (speed == B110)		printf("110 baud\n");
	else if (speed == B134)		printf("134 baud\n");
	else if (speed == B150)		printf("150 baud\n");
	else if (speed == B200)		printf("200 baud\n");
	else if (speed == B300)		printf("300 baud\n");
	else if (speed == B600)		printf("600 baud\n");
	else if (speed == B1200)	printf("1200 baud\n");
	else if (speed == B1800)	printf("1800 baud\n");
	else if (speed == B2400)	printf("2400 baud\n");
	else if (speed == B4800)	printf("4800 baud\n");
	else if (speed == B9600)	printf("9600 baud\n");
	else if (speed == B19200)	printf("19200 baud\n");
	else if (speed == B38400)	printf("38400 baud\n");
	else			printf("unknown speed\n");

	speed = cfgetospeed(&term);
	printf("tty line output speed is ");

	if	(speed == B50)		printf("50 baud\n");
	else if (speed == B75)		printf("75 baud\n");
	else if (speed == B110)		printf("110 baud\n");
	else if (speed == B134)		printf("134 baud\n");
	else if (speed == B150)		printf("150 baud\n");
	else if (speed == B200)		printf("200 baud\n");
	else if (speed == B300)		printf("300 baud\n");
	else if (speed == B600)		printf("600 baud\n");
	else if (speed == B1200)	printf("1200 baud\n");
	else if (speed == B1800)	printf("1800 baud\n");
	else if (speed == B2400)	printf("2400 baud\n");
	else if (speed == B4800)	printf("4800 baud\n");
	else if (speed == B9600)	printf("9600 baud\n");
	else if (speed == B19200)	printf("19200 baud\n");
	else if (speed == B38400)	printf("38400 baud\n");
	else			printf("unknown speed\n");

	exit(EXIT_SUCCESS);
}
```

See Also

cfgetispeed(), cfgetospeed(), cfsetispeed(), cfsetospeed(), Programming COHERENT, tcdrain(), tcflow(), tcflush(), tcgetattr(), tcsendbreak(), tcsetattr(), termio, termios.h

Notes

If a program that uses **termios** has set the **termio** flag **ISIG** (which enables signals) and receives character **SUSP** (normally **<ctrl-Z>**), it sends the signal **SIGTSTP** to the current process group. By default, **termios** then discards **SUSP**. Character **SUSP**, as its name implies, tells a program to suspend operation and recede into the background. Please note that because COHERENT does not yet support job control, **SUSP** at present will do nothing.

termios.h — Header File

Definitions used with POSIX extended terminal interface
#include <termios.h>

Header file **<termios.h>** defines the structures and macros that implement the POSIX Standard's extensions to the **termio** interface.

See Also

header files, termios

test — Command

Evaluate conditional expression
test *expression ...*

test evaluates an *expression*, which consists of string comparisons, numerical comparisons, and tests of file attributes. For example, a **test** command might be used within a shell script to test whether a certain file exists and is readable.

The logical result (true or false) of the *expression* is returned by the command for use by another shell construct, such as the command **if**.

Under the Korn shell, **test** is a built-in command that returns zero if *expression* is true, and one if it is false. Under the Bourne shell, **test** is not a built-in command; rather, the Bourne shell uses the command **/bin/test** to test expressions. **/bin/sh** returns zero if the expression is true, one if it is false, and two if a syntax error (or other error) occurred.

Expression Options

test recognizes the following options, one or more of which can be built into an *expression*:

! *exp*	Negates the logical value of expression *exp*.
string1 **!=** *string2*	*string1* is not equal to *string2*.
string1 **<** *string2*	*string1* is lexicographically less than *string2* (**sh** only).
string1 **=** *string2*	*string1* is equal to *string2*.
string1 **>** *string2*	*string1* is lexicographically greater than *string2* (**sh** only).
(*exp***)**	Parentheses allow expression grouping.
exp1 **-a** *exp2*	Both expressions *exp1* and *exp2* are true.
-b *file*	*file* is a block-special device.
-c *file*	*file* is a character-special file.
-d *file*	*file* exists and is a directory.
-e *file*	*file* exists (**/bin/test** only).
file1 **-ef** *file2*	*file1* is the same file as *file2*.
n1 **-eq** *n2*	Numbers *n1* and *n2* are equal. Please note that **test** evalutes the expression as zero. Thus, if one of the arguments is a variable that is not set, **test** treats it as if it were zero. For example, consider the expression: `if [ "$notset" -eq 0 ]` If **notset** is not set, **test** evaluates it to zero and so returns **true**.
-f *file*	*file* exists and is an ordinary file.
-g *file*	File mode has **setgid** bit.
n1 **-ge** *n2*	Number *n1* is greater than or equal to *n2.*
n1 **-gt** *n2*	Number *n1* is greater than *n2.*
-k *file*	File mode has sticky bit.
-L *file*	File is a symbolic link.
n1 **-le** *n2*	Number *n1* is less than or equal to *n2.*
n1 **-lt** *n2*	Number *n1* is less than *n2.*
-n *string*	*string* has nonzero length.
n1 **-ne** *n2*	Numbers *n1* and *n2* are not equal.
file1 **-nt** *file2*	*file1* is newer than *file2*.
exp1 **-o** *exp2*	Either expression *exp1* or *exp2* is true. **-a** has greater precedence than **-o**.
file1 **-ot** *file2*	*file1* is older than *file2*.
-p *file*	*file* is a named pipe.
-r *file*	*file* exists and is readable.
-s *file*	*file* exists and has nonzero size.
-t *[fd]*	*fd* is the file descriptor number of a file that is open and a terminal. The Bourne shell requires that *fd* be given; under the Korn shell, however, defaults to the standard output (file descriptor 1) if no *fd* is given.
-u *file*	File mode has **setuid** set.
-w *file*	*file* exists and is writable.
-x *file*	*file* exists and executable.
-z *string*	*string* has zero length (is a null string).
string	*string* has nonzero length.

Implementations of test

The implementation of **test** under the Bourne shell has been rewritten for COHERENT release 4.2, both to extend its range of features and to make it more compliant with published standards. Although this makes **test** more useful to programmers, it may create problems when you try to execute a Bourne-shell script written under COHERENT release 4.2 on an earlier release of COHERENT. The following describes how the Bourne shell's implementation of **test** was designed; and how it differs both from earlier implementations under the Bourne shell and from the implementation under the Korn shell.

To begin, the Bourne shell's implementation of **test** attempts to comply with the POSIX Standard, comply with previous COHERENT releases of **test**, and comply with System-V UNIX to the greatest extent possible. However, these objectives are mutually exclusive. See the POSIX Standard P1003.2/D11.2 §4.62, especially the Rationale, for details of some of the problems. In particular, System V and Berkeley differ in the way they parse some expressions, which leads the POSIX Standard to specify test behavior for a minimal set of expressions, *not* including **-a** and **-o**.

The following details differences among the various implementations of **test**. First, the following options were not implemented in the Bourne shell's implementation of **test** prior to COHERENT release 4.2, but were included in the Korn shell's implementation: **-b**, **-c**, **-ef**, **-g**, **-k**, **-L**, **-nt**, **-ot**, **-p**, **-u**, and **-x**. Of these, the following are not described in the POSIX Standard: **-k**, **-L**, **-ef**, **-nt**, and **-ot**. Note that Bourne-shell scripts that use any of the above options to **test** will *not* run on versions of COHERENT prior to release 4.2, but will run under the Korn shell.

Next, the Bourne shell for COHERENT 4.2 implements the POSIX Standard's option **-e** and the options **<** and **>**. Bourne-shell scripts that use any of these three options to **test** will *not* run on versions of COHERENT prior to release 4.2, nor will they run under the Korn shell.

The definitions of the options **-f** and **-t** have been changed from the Berkeley standard to that described in the POSIX Standard. Berkeley defines **-f** as meaning that a file exists and is not a directory; whereas the POSIX Standard defines it as meaning that a file exists and is a regular file. Versions of the Bourne shell prior to COHERENT 4.2 use the Berkeley definitions; whereas all version of the Korn shell and Bourne shell under COHERENT 4.2 use the POSIX Standard's definition. Berkeley gives **-t** a default value of one if it is not used with an argument; whereas the POSIX Standard requires that **-t** have an argument. The Korn shell and all versions of the Bourne shell prior to COHERENT 4.2 use the Berkeley definition; whereas the Bourne shell under COHERENT 4.2 uses the POSIX Standard's definition. These differences are subtle, but important. Thus, a Bourne shell script that uses either of these options may not run correctly when imported into COHERENT 4.2 from earlier versions of COHERENT, or when exported from COHERENT 4.2 to them or to the Korn shell.

Finally, **test** under the Korn shell and under the Bourne shell prior to COHERENT 4.2 returns zero if an expression is true and one either if the expression is false or if the expression contained a syntax error. However, **test** under the Bourne shell for COHERENT 4.2 returns zero if an expression is true, one if it is false, and two if a syntax error occurred. Bourne-shell scripts that pay close attention to what **test** returns may not run correctly when imported into COHERENT 4.2 from earlier implementations of COHERENT, or when exported from COHERENT 4.2 to earlier versions of COHERENT or to the Korn shell.

Example

The following example uses the **test** command to determine whether a file is writable.

```
if test ! -w /dev/lp
then
      echo The line printer is inaccessible.
fi
```

Under COHERENT, the command '[' is linked to **test**. If invoked as '[', **test** checks that its last argument is ']'. This allows an alternative syntax: simply enclose *expression* in square brackets. For example, the above example can be written as follows:

```
if [ ! -w /dev/lp ]
then
      echo The line printer is inaccessible.
fi
```

For a more extended example of the square-bracket syntax, see **sh**.

See Also

commands, expr, find, if, ksh, sh, while

LEXICON

Notes

The Korn shell's version of this command is based on the public-domain version written by Erik Baalbergen and Arnold Robbins.

tgetent() — termcap Function (libterm)

Read termcap entry
#include <curses.h>
#include <term.h>
int tgetent(*bp, name***)**
char **bp, *name***;*

tgetent() is one of a set of functions that read a **termcap** terminal description. It extracts the entry from file **/etc/termcap** for the terminal *name* and writes it into a buffer at address *bp*. *bp* should be a character buffer of 1,024 bytes and must be retained through all subsequent calls to the other functions. It returns **-1** if it cannot open **/etc/termcap**, zero if the terminal name given does not have an entry, and one upon a successful search.

tgetent() first looks in the environment to see if the **termcap** variable had already been set. If it finds that the variable **termcap** has been set, that the value does *not* begin with a slash, and that the terminal type name in the **termcap** variable is the same as that in the environment variable **TERM**, then **tgetent()** uses the **termcap** string instead of reading the file **/etc/termcap**. However, if the **termcap** string does begin with a slash, then it is used as the path name of a terminal-capabilities file other than **/etc/termcap.** This can speed entry into programs that call **tgetent()**, and can be used to help debug new terminal descriptions.

Files

/etc/termcap — Terminal capabilities data base
/usr/lib/libterm.a — Function library

See Also

termcap

tgetflag() — termcap Function

Get termcap Boolean entry
#include <curses.h>
#include <term.h>
int tgetflag(*name***)**
char **name*;

tgetflag() is one of a set of functions that read a **termcap** terminal description. It returns one if the requested Boolean capability *name* is present in the terminal's **termcap** entry, zero if it is not.

Files

/etc/termcap — Terminal capabilities data base
/usr/lib/libterm.a — Function library

See Also

termcap

tgetnum() — termcap Function (libterm)

Get termcap numeric feature
#include <curses.h>
#include <term.h>
int tgetnum(*name***)**
char **name*;*

tgetnum() is one of a set of functions that read a **termcap** terminal description. It returns the value of the numeric feature *name*, as defined in the terminal's **termcap** entry. It returns **-1** if the feature is not present in the terminal's entry.

Files

/etc/termcap — Terminal capabilities data base
/usr/lib/libterm.a — Function library

See Also

termcap

tgetstr() — termcap Function (libterm)

Get termcap string entry
#include <curses.h>
#include <term.h>
char ***tgetstr(***name, area***)***
char **name***, ***area***;**

tgetstr() is one of a set of functions that read a **termcap** terminal description. It reads the string value of feature *name* from the terminal's **termcap** description, and writes it into the buffer at address *area*. It also advances the value of the pointer to *area*.

tgetstr() decodes the abbreviations for the fields used in the **termcap** entry, except for padding and for cursor-addressing information.

Files

/etc/termcap — Terminal capabilities data base
/usr/lib/libterm.a — Function library

See Also

termcap

tgoto() — termcap Function (libterm)

Read/interpret termcap cursor-addressing string
#include <curses.h>
#include <term.h>
char ***tgoto(***cm, destcol, destline***)***
char **cm***; **int** scrcol***,*** scrline***;***

tgoto() is one of a set of functions that read a **termcap** terminal description. It decodes a cursor-addressing string from the *cm* **termcap** feature, and writes it onto the screen, at column *scrcol* and line *destline*. **tgoto()** uses the external variables **UP** (from the **up** feature) and **BC** (if **bc** is given rather than **bs**) if it is necessary to avoid placing **\n, <ctrl-D>**, or **<ctrl-@>** into the returned string. Programs calling **tgoto()** should turn off the **XTABS** bits, as **tgoto()** may write a tab. If a '%' sequence is given that is not understood, **tgoto()** returns "OOPS".

Files

/etc/termcap — Terminal capabilities data base
/usr/lib/libterm.a — Function library

See Also

termcap

tic — Command

Compile a terminfo description
tic [-v[*n***]]** *sourcefile*

The command **tic** compiles a *sourcefile* of **terminfo** information into a binary object.

sourcefile must be self-contained, i.e., it may not contain "use" entries that refer to terminals not described fully in the same file.

The object files generated by **tic** are normally placed into subdirectories of the directory **/usr/lib/terminfo**. If the environment variable **TERMINFO** is defined, it is assumed to name an alternative directory to use.

The flag **-v***n* tells **tic** to output debugging and tracing information. *n* sets the amount of debugging information to produce, as follows:

1 Names of files created
2 Information related to the "use" facility
3 Statistics from the hashing algorithm
5 String-table memory allocations
7 Entries into the string-table
8 List of tokens encountered by scanner
9 All values computed in construction of the hash table

n is set to one by default.

Files

/usr/lib/terminfo/* — Default location of object files

See Also

commands, infocmp, terminfo, term
Strang, J., Mui, L., O'Reilly, T.: *termcap and terminfo*. Sebastopol, CA: O'Reilly & Associates, Inc., 1991.

Notes

tic was written by Pavel Curtis of Cornell University. It was ported to COHERENT by Udo Munk.

time — Overview

COHERENT includes a number of routines that allow you to set and manipulate time, as recorded on the system's clock, into a variety of formats. These routines should be adequate for nearly any task that involves temporal calculations or the maintenance of data gathered over a long period of time.

All functions, global variables, and manifest constants used in connection with time are defined and described in the header files **time.h** and **timeb.h**. In brief, time manipulates two data elements: the type **time_t**, and the structure **tm**.

time_t is defined in the header file **<time.h>**. COHERENT follows the UNIX standard and initializes **time_t** to the number of seconds since January 1, 1970, 0h00m00s GMT; this moment, in turn, is rendered as day 2,440,587.5 on the Julian calendar. This allows accurate calculation of time as far back as January 1, 4713 B.C.

The structure **tm** is defined in the header file **<time.h>**. It is defined as follows:

```
struct tm {
        int tm_sec;             /* current time, seconds */
        int tm_min;             /* current time, minutes */
        int tm_hour;            /* current time, hour */
        int tm_mday;            /* day of the month, 1-31 */
        int tm_mon;             /* month, 1-12 */
        int tm_year;            /* year since 1900 */
        int tm_wday;            /* day of the week, Sunday = 0 */
        int tm_yday;            /* day in the current year */
        int tm_isdst;           /* is daylight-savings time now in effect? */
};
```

For an example of manipulating this structure in a C program, see the Lexicon entry for **localtime()**.

Standard Time Functions

The COHERENT system includes the following functions to manipulate time:

asctime() . . . . . Convert time structure to ASCII string
clock() . . . . . Get processor time
ctime() . . . . . Convert system time to an ASCII string
difftime() . . . . . Return difference between two times
gmtime() . . . . . Convert system time to calendar structure
localtime() . . . . . Convert system time to calendar structure
mktime() . . . . . Turn broken-down time into calendar time
strftime() . . . . . Format locale-specific time
time() . . . . . Get the current time
tzset() . . . . . Set local time zone

To print out the local time, a program must perform the following tasks:

1. Read the system time with **time()**. This function returns a **time_t**.
2. Pass the **time_t** returned by **time()** to the function **localtime()**. This function breaks it down into the **tm** structure,
3. Pass **localtime()**'s output to **asctime()**, which transforms the **tm** structure into an ASCII string.
4. Finally, pass the output of **asctime()** to **printf()**, to displays it on the standard output device.

See the entry for **asctime()** for an example C program that goes through the above steps.

Special Time Functions

COHERENT includes a number of extensions to the time functions used by standard UNIX systems. These are designed to increase the scope and accuracy of time-handling, and to ease calculation of some time elements.

COHERENT holds information about your time locale in the environmental variable **TIMEZONE**. This variable is described in detail in its Lexicon article. In brief, it consists of seven fields:

1. Name of the local standard time zone
2. Offset from Greenwich Mean Time, in minutes
3. Name of the local daylight time zone
4. Date when daylight-savings time begins
5. Date when daylight-savings time ends
6. Hour when daylight-savings time begins
7. Hour when daylight-savings time ends

The fields are defined in such a way that any form of daylight-saving adjustment can be handled correctly. For example, the United States shifts into daylight-savings time on the first Sunday in April; whereas Brazil shifts into daylight-savings time on a set day each spring.

The function **tzset()** parses **TIMEZONE** into the following external variables:

timezone Seconds from GMT to give local time
tzname[2][16] Character array of names of standard and daylight times

For details on manipulations these variable, see the Lexicon entry for **tzset()**. The library **libmisc.a** contains the following functions that convert time from Julian to Gregorian form:

time_to_jday() Convert **time_t** to the Julian date
jday_to_time() Convert Julian date to **time_t**
tm_to_jday() Convert **tm** structure to Julian date
jday_to_tm() Convert Julian date to **tm** structure

COHERENT's time functions assume that conversion to the Gregorian calendar occurred October 1582, when it was first adopted in Rome. However, various nations adopted the Gregorian calendar at different times; for example, it was adopted in the British Empire (including its American colonies) only in September 1752. (This, by the way, is the date assumed by the COHERENT command **cal**, as you would see if you typed the command **cal 9 1752**.) Users in northern and eastern Europe, and in European-influenced areas of Asia (e.g., India) may wish to to write their own functions to convert historical data properly from the Julian to the Gregorian calendar.

Example

For an example of some time functions, see the entry for **asctime()**.

See Also

cal, libc, libmisc

Notes

COHERENT also includes the system call **ftime()**, which returns the current system time. Because the ANSI Standard eliminates **ftime()**, users are urged to replace this system call with calls to **time()**.

UNIX System V defines **time_t** in header file **<sys/types.h>**, whereas COHERENT defines it in **time.h**. This should not affect the porting of programs from UNIX to COHERENT, but it may affect the porting of programs in the other direction.

time — Command

Time the execution of a command
time [*command***]**

time invokes the given *command* with any arguments provided. Upon termination, **time** prints the elapsed real time, CPU time in the system, and CPU time in the user program on the standard error output.

See Also

commands, date, ps, times

Diagnostics

If the *command* terminates abnormally, **time** displays an error message that explains why.

time.h — Header File

Give time-description structure
#include <time.h>

The header file **time.h** prototypes the routines that COHERENT uses to manipulate time, and declares the constants and data types they use.

See Also

header files, time [overview]
ANSI Standard, §7.12

time() — System Call (libc)

Get current system time
#include <time.h>
time_t time(*tp***)**
time_t ****tp*;**

time() reads and returns the current system time. COHERENT defines the current system time as the number of seconds since January 1, 1970, 0h00m00s GMT.

tp points to a data element of the type **time_t**, which the header file **time.h** defines as being equivalent to a **long**. If *tp* is initialized to a value other than NULL, then **time()** attempts to write the system time into the address to which *tp* points. If, however, *tp* is initialized to NULL, then **time()** returns the current system time but does not attempt to write it anywhere.

Example

For an example of this call, see the entry for **asctime()**.

See Also

date, libc, time [overview], time.h
ANSI Standard, §7.12.2.4
POSIX Standard, §4.5.1

Notes

UNIX System V defines **time_t** in header file **<sys/types.h>**, whereas COHERENT defines it in **time.h**. This should not affect the porting of programs from UNIX to COHERENT, but it may affect the porting of programs in the other direction.

timeb.h — Header File

Define timeb structure
#include <sys/timeb.h>

The header file **timeb.h** defines the structure **timeb**, which is used by the function **ftime()** to return time information.

See Also

ftime(), header files, time [overview]

timeout.h — Header File

Define the timer queue
#include <timeout.h>

timeout.h defines the timeout queue. The timeout queue can, as its name implies, be used to call a function when a process has "timed out".

See Also

header files

Notes

This header file is obsolete, and will be dropped from a future release of COHERENT. Its use is strongly discouraged.

times — Command

Print total user and system times
times

times prints the total elapsed user time and system time for the current shell and all its children. It gives each time in minutes, seconds and tenths of seconds. For example,

```
1m11.8s 1m35.8s
```

indicates a total user time of 1 minute 11.8 seconds, and a total system time of 1 minute 35.8 seconds.

The shell executes **times** directly.

See Also

commands, ksh, time, sh

times.h — Header File

Definitions used with times() system call
#include <sys/times.h>

times.h defines the structure **tms**, which is used by the system call **times()**.

See Also

header files, times()
POSIX Standard, §4.5.2

times() — System Call (libc)

Obtain process execution times
#include <sys/times.h>
#include <time.h>
int times(*tbp***)**
struct tms **tbp***;*

times() reads CPU time information about the current process and its children, and writes it into the structure pointed to by *tbp*. The structure **tms** is declared in the header file **sys/times.h**, as follows:

```
struct tms {
        clock_t tms_utime;                  /* process user time */
        clock_t tms_stime;                  /* process system time */
        clock_t tms_cutime;                 /* childrens' user times */
        clock_t tms_cstime;                 /* childrens' system times */
};
```

All of the times are measured in basic machine cycles, or **CLK_TCK**.

The childrens' times include the sum of the times of all terminated child processes of the current process and of all of their children. The *user* time represents execution time of user code, whereas *system* time represents system overhead, such as executing system calls, processing signals, and other monitoring functions.

times() returns the number of ticks that have passed since system startup.

Files

<sys/times.h>
<time.h>

See Also

acct(), ftime(), libc, time() times.h,
POSIX Standard, §4.5.2

TIMEZONE — Environmental Variable

Time zone information
TIMEZONE=*standard***:***offset***[:***daylight***: ***date***:***date***:***hour***:***minutes***]**

The COHERENT system records time internally as Greenwich Mean Time (GMT). It does so to make it easier to coordinate exchange of information across systems in different time zones around the world.

TIMEZONE is an environmental parameter that holds information about your local time zone. This information is used by COHERENT's time routines to convert GMT to the date and time in your local area. **TIMEZONE** takes into account your local time zone's offset from Greenwich, whether your country uses daylight savings time, and the date and hour that daylight savings time begins and ends.

To set **TIMEZONE**, use the command

export TIMEZONE=[*description***]**

where *description* is the string that describes your time zone. What this string consists of will be described below. Most users write this command into the file **.profile**, so that **TIMEZONE** is set automatically whenever they log onto the COHERENT system.

COHERENT's installation procedure creates file **/etc/timezone**, which sets **TIMEZONE**. This file is executed by **/etc/profile** when each user logs in. Thus, you must set the **TIMEZONE** in your **.profile** only if it differs from the system's **TIMEZONE** as set in **/etc/timezone**. This would be necessary if, for example, a user in New York were to regularly login on a system in Chicago.

The Description String

A **TIMEZONE** description string consists of seven fields that are separated by colons. Fields 1 and 2 must be filled; fields 3 through 7 are optional.

Field 1 gives the name of your standard time zone. Field 2 gives the time zone's offset from Greenwich Mean Time in minutes. Offsets are positive for time zones west of Greenwich and negative for time zones east of Greenwich. For example, users in Chicago set these fields as follows:

```
TIMEZONE=CST:360
```

CST is an abbreviation for Central Standard Time, that area's time zone; and 360 refers to the fact that Chicago's time zone is 360 minutes (six hours) ahead of (that is, earlier than) Greenwich.

Field 3 gives the name of the local daylight saving time zone. In Chicago, for example, this field would be set as follows:

```
TIMEZONE=CST:360:CDT
```

CDT is an abbreviation for Central Daylight Time. The absence of this field indicates that your area does not use daylight saving time.

Fields 4 and 5 specify the dates on which daylight saving time begins and ends. If field 3 is set but fields 4 and 5 are not, changes between standard time and daylight saving time are assumed to occur at the times legislated in the United States: at 2 A.M. standard time on the first Sunday in April, and at 2 A.M. daylight saving time on the last Sunday in October.

Fields 4 and 5 each consist of three numbers separated by periods. The first number specifies which occurrence of the day in the month marks the change, counting positive occurrences from the beginning of the month and negative occurrences from the the end of the month. The second number specifies a day of the week, numbering Sunday as one. The third number specifies a month of the year, numbering January as one. For example, in Chicago fields 4 and 5 are set to the following:

```
TIMEZONE=CST:360:CDT:1.1.4:-1.1.10
```

If the first number in either field is set to zero, then the last two numbers are assumed to indicate an absolute date. This is done because some countries switch to daylight saving time on the same day each year, instead of a given day of the week.

Finally, fields 6 and 7 specify the hour of the day at which daylight saving time begins and ends, and the number of minutes of adjustment. In Chicago, these are set as follows:

```
TIMEZONE=CST:360:CDT:1.1.4:-1.1.10:2:60
```

The '2' of field 6 indicates that the switch to daylight savings time occurs at 2 A.M. The "60" of field 7 indicates that daylight savings time changes the local time by 60 minutes. Although 60 minutes is the standard change, some regions of the world shift by 30, 45, 90, or 120 minutes; the last shift is also called "double daylight saving time".

For an example of this variable's use in a program, see the entry for **asctime()**.

See Also

environmental variables, time [overview]

Notes

File **/etc/default/login** defines **TIMEZONE** differently: it uses the same format as the COHERENT environmental variable **TZ**, which is set in file **/etc/timezone**. Note that **TZ** and **TIMEZONE** as defined in **/etc/default/login** must be identical, or much confusion will result.

For those requiring more information on this subject, much research has been performed by astrologers. See *Time Changes in the World*, compiled by Doris Chase Doane (three volumes, Hollywood, California, Professional Astrologers, Inc., 1970).

TMPDIR — Environmental Variable

Directory that holds temporary files

The command **cc** reads the environmental variable **TMPDIR** to see where you want it to write its temporary files. You can speed compilation by building a RAM disk and pointing **TMPDIR** to point at it.

For example, if you have created a RAM disk and mounted it as **/z**, then by embedding the instruction

```
export TMPDIR=/z/tmp
```

in your **.profile**, you can ensure that **cc** will write all of its temporary files onto the RAM disk.

See Also

cc, environmental variables, ram

tmpfile() — STDIO Function (libc)

Create a temporary file
#include <stdio.h>
FILE *tmpfile(void);

The function **tmpfile()** creates a file to hold data temporarily. The file is opened into binary update mode (**wb+**) and is removed automatically when it is closed or when the program exits. There is no way to access the temporary file by name. If your program needs to do so, it should open a file explicitly.

tmpfile() returns NULL if it could not create a temporary file. If it could, it returns a pointer to the **FILE** associated with the temporary file. The function **exit()** removes all files created by **tmpfile()**.

Example

This example implements a primitive file editor that can edit large files. It uses two temporary files to keep all changes. The editor accepts the following commands:

dn	delete; **d52** deletes line 52
in	insert; **i7** inserts line before line 7
pn	print; **p17** prints line 17
p	print the entire file
w	write the edited file and quit
q	quit without writing the file

The entire temporary file is copied with each command.

```
#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>
#include <stdarg.h>

FILE *fp, *tmp[2];
int linecount;

fatal(message)
char *message;
{
      fprintf(stderr, "%s\n", message);
      exit(EXIT_FAILURE);
}

/*
 * Copy up to line number or EOF.
 * Return number of lines copied.
 */
static int
copy(line, *ifp, ofp)
int line; FILE *ifp, *ofp;
{
      int i, c, count;

      count = 0;
      for(c=i=1; (i<line || line==-1) && c!=EOF; i++) {
            while((c = fgetc(ifp)) != EOF && c != '\n')
                  fputc(c, ofp);

            if(c == '\n') {
                  count++;
                  fputc('\n', ofp);
            }
      }
      return(count);
}

/*
 * Read a file until line number is read.
 * Return 1 if line is found before EOF.
 */
static int
find(line, ifp)
int line; FILE *ifp;
{
      int i, c;

      for(c=i=1; i<line && c!=EOF; i++)
            while((c = fgetc(ifp)) != EOF && c != '\n')
                  ;
      return(c != EOF);
}

main(int argc, char *argv[])
{
      int i, line, args;
      char c, cmdbuf[80];

      if(argc != 2)
            fatal("usage: tmpfile filename\n");

      if((tmp[0]=tmpfile())==NULL||(tmp[1]=tmpfile())==NULL)
            fatal("Error opening tmpfile\n");

      if((fp = fopen(argv[1], "r")) == NULL)
            fatal("Error opening input file\n");
```

```
linecount = copy(-1, fp, tmp[i = 0]);
fclose(fp);

/* one file pass per command */
for(;;) {
      if(gets(cmdbuf) == NULL)
            fatal("EOF on stdin\n");

      if(!(args = sscanf(cmdbuf, "%c%d", &c, &line)))
            continue;
      fseek(tmp[i], 0L, SEEK_SET);

      switch(c) {
      /* Write edited file */
      case 'w':
            if((fp = fopen(argv[1], "w")) ==  NULL)
                  fatal("Error opening file\n");
            copy(linecount + 1, tmp[i], fp);
            fclose(fp);

      /* Quit */
      case 'q':
            exit(EXIT_SUCCESS);

      /* Print entire file */
      case 'p':
            if(args == 1) {
                  copy(linecount + 1, tmp[i], stdout);
                  continue;
            }
            if(find(line, tmp[i]))
                  copy(2, tmp[i], stdout);
            continue;

      /* Delete a line */
      case 'd':
            if(args == 1)
                  printf("dn where n is a number\n");
            else if(line > linecount)
                  printf("only %d lines\n", linecount);

            else {
                  copy(line, tmp[i], tmp[i^1]);
                  if(find(2, tmp[i]))
                        copy(-1, tmp[i], tmp[i^1]);

                  linecount--;
                  fseek(tmp[i], 0L, SEEK_SET);
                  i ^= 1;
            }
            continue;

      /* Insert a line */
      case 'i':
            if(1 == args)
                  printf("in where n is a number\n");
            else if(line > linecount)
                  printf("only %d lines\n", linecount);

            else {
                  copy(line, tmp[i], tmp[i^1]);
                  printf("Enter inserted line\n");
                  copy(2, stdin, tmp[i^1]);
                  copy(-1, tmp[i], tmp[i^1]);
                  linecount++;

                  fseek(tmp[i], 0L, SEEK_SET);
                  i ^= 1;
            }
            continue;
```

```
            default:
                  printf("Invalid request\n");
                  continue;
            }
      }
}
```

See Also

mktemp(), libc, tempnam(), tmpnam()
ANSI Standard, §7.9.4.3
POSIX Standard, §8.1

Notes

If a program exits abnormally or aborts, the files created by **tmpfile()** may not be removed.

tmpnam() — STDIO Function (libc)

Generate a unique name for a temporary file
#include <stdio.h>
char *tmpnam(*name***);**
char ****name*;**

tmpnam() constructs a unique name for a file. The names returned by **tmpnam()** generally are mechanical concatenations of letters, and therefore are mostly used to name temporary files, which are never seen by the user. A file named by **tmpnam()** does not automatically disappear when the program exits. You must explicitly remove it before the program ends if you want it to disappear.

name points to the buffer into which **tmpnam()** writes the name it generates. If *name* is set to NULL, **tmpnam()** writes the name into an internal buffer that may be overwritten each time you call this function.

tmpnam() returns a pointer to the temporary name. Unlike the related function **tempnam()**, **tmpnam()** assumes that the temporary file will be written into directory **/tmp** and builds the name accordingly.

Example

For an example of this function, see **execve()**.

See Also

libc, mktemp(), tempnam()
ANSI Standard, §7.9.4.4
POSIX Standard, §8.1

Notes

If you want the file name to be written into *buffer*, you should allocate at least **L_tmpnam** bytes of memory for it; **L_tmpnam** is defined in the header **stdio.h**. Under COHERENT, it is 64 characters long.

toascii() — ctype Function (libc)

Convert characters to ASCII
#include <ctype.h>
int toascii(*c***)** int *c***;**

The function **toascii()** takes the integer value *c*, keeps the low seven bits unchanged, and changes the others to zero. This, in effect, transforms the integer value to an ASCII character. **toascii()** then returns the transformed integer. If *c* is already a valid ASCII character, **toascii()** returns it unchanged.

Example

This example prompts for a file name. It then opens the file and prints its contents, while converting all non-alphanumeric characters to alphanumeric.

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
```

```
main()
{
      FILE *fp;
      int ch;
      int filename[20];

      printf("Enter file name: ");
      fflush(stdout);
      gets(filename);

      if ((fp = fopen(filename, "r")) != NULL) {
            while ((ch = fgetc(fp)) != EOF)
                  putchar(isascii(ch) ? ch : toascii(ch));
      } else {
            printf("Cannot open %s\n", filename);
            exit(EXIT_FAILURE);
      }
      return(EXIT_SUCCESS);
}
```

See Also

isascii(), libc

Notes

This function is not described in the ANSI Standard. Any program that uses it does not conform strictly to the Standard, and may not be portable to other compilers or environments.

tolower() — ctype Function (libc)

Convert characters to lower case
#include <ctype.h>
int tolower(*c***) int** *c***;**

The function **tolower()** converts the character *c* to lower case. It returns *c* converted to lower case. If *c* is not upper-case character, that is, a character for which **isupper()** returns true, **toupper()** returns it unchanged.

Example

The following example demonstrates **tolower()** and **toupper()**. It reverses the case of every character in a text file.

```
#include <ctype.h>
#include <stdio.h>

main()
{
      FILE *fp;
      int ch;
      int filename[100];

      printf("Enter name of file to use: ");
      fflush(stdout);
      gets(filename);

      if ((fp = fopen(filename,"r")) != NULL) {
            while ((ch = fgetc(fp)) != EOF) {
                  if (islower(ch))
                        putchar(toupper(ch));
                  else if (isupper(ch))
                        putchar(tolower(ch));
                  else
                        putchar(ch);
            }
      } else
             printf("Cannot open %s.\n", filename);
}
```

See Also

_tolower(), libc, toupper()
ANSI Standard, §7.3.2.1
POSIX Standard, §8.1

touch — Command

Update modification time of a file
touch [-c] *file ...*

COHERENT keeps track of when each file was last modified. **touch** changes the modification time of each *file* to the current time, but does not modify its contents. By default, **touch** creates *file* if it does not already exist; the **-c** flag suppresses this.

See Also

commands, make

toupper() — ctype Function (libc)

Convert characters to upper case
#include <ctype.h>
int toupper(*c***) int** *c***;**

toupper() converts the letter *c* to upper case and returns the converted character. If *c* is not an lower-case character, that is, any character for which **islower()** returns true, **toupper()** returns it unchanged.

Example

For an example of this routine, see the entry for **tolower()**.

See Also

_toupper(), libc, tolower()
ANSI Standard, §7.3.2.2
POSIX Standard, §8.1

tparm() — terminfo Function

Output a parameterized string
#include <curses.h>
tparm(*string***,***p1...p9***)**
char **string***, parm1 ... par9***;**

COHERENT comes with a set of functions that let you use **terminfo** descriptions to manipulate a terminal. **tparm()** outputs a parameterized string.

A *parameterized string* is a string into which parameters can be inserted, as in a **printf()** formatting string. Under **terminfo**, a parameterized string can hold up to nine parameters. **tparm()** expands the parameters, inserts them into the appropriate "slots" within the string, and then outputs the string.

See the Lexicon entry on **terminfo** for more information on parameterized strings.

See Also

curses.h, terminfo, tputs()

tputs() — termcap/terminfo Function (libterm/libcurses)

Read/decode leading padding information
#include <curses.h>
#include <term.h>
tputs(*name, affcnt, outc***)**
char **name***;* **int** *affcnt***; int (***outc***)();**

tputs() is one of a set of functions that read a **termcap** or **terminfo** terminal description.

tputs() decodes the leading padding information of the string *name*. When you use **tputs()** to interpret the **terminfo** data base, *name* should point to a string that names one of **terminfo**'s variables, as defined in the Lexicon entry for **terminfo**; e.g., **auto_right_margin** or **auto_left_margin**. When you use **tputs()** to interpret the **termcap** data base, *name* should point to **termcap**'s variable code, e.g., **am**.

affcnt gives the number of lines affected by the operation. Set it to one if it is not applicable.

outc points to the routine that writes each character.

Files

/etc/termcap — Terminal capabilities data base
/etc/terminfo — Terminal capabilities data base
/usr/lib/libcurses.a — Routines for reading **terminfo** descriptions
/usr/lib/libterm.a — Routines for reading **termcap** descriptions

See Also

libcurses, libterm, termcap, terminfo

Notes

As noted above, **tputs()** can read either a **termcap** or a **terminfo** description. The **termcap** version of **tputs()** lives in library **/usr/lib/libterm.a**. To obtain the **termcap** version of **tputs()**, link in the library **/usr/lib/libterm.a**. To obtain the **terminfo** version, however, link in the library **/usr/lib/libcurses.a**.

tr — Command

Translate characters
tr [-cds] *string1* **[***string2***]**

tr reads characters from the standard input, possibly translates each to another value or deletes it, and writes to standard output.

Each specified *string* may contain literal characters of the form *a* or *\b* (where *b* is non-numeric), octal representations of the form *\ooo* (where *o* is an octal digit), and character ranges of the form *X-Y*. **tr** rewrites each *string* with the appropriate conversions and range expansions.

If an input character is in *string1*, **tr** outputs the corresponding character of *string2*. If *string2* is shorter than *string1*, the result is the last character in *string2.*

The following flags control how **tr** translates characters:

-c Replace *string1* by the set of characters not in *string1.*

-d Delete characters in *string1* rather than translating them.

-s The "squeeze" option: map a sequence of the same character from *string1* to one output character.

Example

The following example prints all sequences of four or more spaces or printing characters from **infile**:

```
tr -cs ' -~' '\12' <infile | grep ....
```

Here *string1* is the range from **<space>** to '~', which includes all printing characters. Because this example uses the flags **-cs**, **tr** maps sequences of nonprinting characters to newline (octal 12).

See Also

ASCII, commands, ctype.h, sed

Notes

Beginning with COHERENT 4.2, the command

```
echo "This is a test." | tr
```

returns

```
This is a test.
```

This behavior does not conform with POSIX Standard, but is required by a number of third-party packages.

tr — Device Driver

Driver to read stored error messages
/dev/trace

The device driver **tr** is the "traceback" driver for the COHERENT kernel. It manipulates an internal buffer that holds error messages from the kernel or another device driver. It has major number 6. This driver is extremely useful to persons who are writing device drivers.

The DDI/DKI kernel routine **cmn_err()** can be invoked by drivers to write formatted messages. By default, it writes

the messages both onto the system's console and into an internal buffer in memory that can hold up to four megabytes of text. Messages that begin with a caret, '^', go to the console but not the internal buffer. Messages that begin with an exclamation point, '!', go to the buffer but not to the console.

The trace driver **tr** reads this internal buffer, and lets you copy its contents into a file for later perusal. This offers two major advantages to persons developing and debugging device drivers:

- First, copious diagnostic output will no longer scroll off the screen and be lost. The trace buffer holds every error message written through **cmn_err()** until you read the buffer. When you install **tr**, you can set the size of the buffer, up to four megabytes.
- Second, if messages are written to the trace buffer only and not to the console, system timing is affected much less than if the messages were written to the console. This makes it easier to catch subtle problems in timing.

To add **tr** to your kernel, do the following:

- Log in as the superuser **root**.
- **cd** to directory **/etc/conf**.
- Execute script **tr/mkdev**. This will walk you through the process of configuring your kernel to use this driver, and create the device **/dev/trace**.
- Execute script **bin/idmkcoh**, to generate a new kernel.
- Invoke the script **/etc/shutdown** to shutdown system, then boot the new kernel.

To read the contents of the trace buffer, simply use the command

```
cp /dev/trace file
```

where *file* is the file into which you wish to copy the contents of the trace buffer.

See Also

device drivers
COHERENT Device Driver Kit: **cmn_err()**

transports — System Administration

Describe mail transportation systems
/usr/lib/mail/transports

The program **smail** reads file **/usr/lib/mail/transports** for information on the commands it can use to deliver mail, either to your local system or to a remote system.

Each entry within **transports** names a transport and sets its attributes. Each entry consists of the following information:

- The name of the transport. This attribute begins the definition of a transport. The name must be unique, it must appear flush with the left margin, and must be followed by a single colon ':'.
- The name of the *driver*, or program that implements the transport. This can be a command that is part of **smail**'s suite of utilities (which are contained in directory **/usr/lib/mail**), or can be an ordinary COHERENT command. If the latter, then the full name of the command that implements the driver is given with a **cmd** attribute. This is demonstrated below.
- A set of generic attributes for the transport. These attributes are "generic" because they can come from a set that can be applied to any router.
- A set of driver-specific attributes. These can be applied only to entries that use this driver.

To extend an entry across multiple lines, begin successive lines with white space.

Attributes of a Transport

The following gives the generic attributes that a transport can have. Each attribute is followed by its type (Boolean, string, or number). To set a string or number attribute, its name should be followed by an '=', then the value to which you are setting it. To set a Boolean attribute, prefix it with a '+'; to unset a Boolean attribute, prefix it with a '-'.

bsmtp (Boolean)
This transport uses a batched SMTP format, in which the message is enclosed within an envelope of SMTP commands. You can use such a transport to send mail in SMTP format to remote hosts, even when direct two-way connections are not feasible. For example, this will work over UUCP and eliminates difficulties with sending arbitrary addresses as arguments to the command **uux**. Use of this attribute also turns on the attribute **dots**. When this attribute is also used with the attribute **uucp**, **smail** uses UUCP-style bang-path addresses in the SMTP envelope.

crlf (Boolean)
If set, each line of the header and message ends within the pair of characters CR:LF rather than a single newline character. In general, this is not a useful attribute, as the SMTP transport (which requires this as a part of the interactive protocol) always does this anyway.

debug (Boolean)
If set, this attribute replaces the body of the message with debugging information. You can use it, for example, as a shadow transport, to watch the flow of mail for debugging purposes. This lets you debug mail while avoid the problems that arise from saving other users' personal correspondence.

dots (Boolean)
If set, then **smail** uses the "hidden-dot" protocol. With this protocol, **smail** prefixes a period '.' onto every line that already begins with a period. All of the various SMTP modes imply this behavior.

driver (string)
This attribute names the specific entity that actually transports the mail. It is required.

error_transport (string)
This attribute names another transport that **smail** can use to send the message, should this transport fail.

from (Boolean)
If set, **smail** supplies a "From<space>" line before the message when it delivers mail via this transport. If this is a remote transport (i.e., the attribute **local** is not turned on), this line ends with the string

remote from *hostname*

where *hostname* is the UUCP name for your local host (as set in file **/etc/uucpname**). This is useful for delivery via UUCP and for delivering mail to standard mailbox files, which require this format.

hbsmtp (Boolean)
"Half-baked" batched SMTP. This is batched SMTP mode without an initial **HELO** command or an ending **QUIT** command. **smail** can use this transport to create files that it will later concatenate into a batch of SMTP commands and multiple messages. Use of this attribute also turns on attribute **dots**.

local_xform (Boolean)
If this attribute is set, **smail** uses the form of the header and envelope information appropriate for delivery to your local host. This changes no existing header field, except that it inserts commas into the fields that name the sender and recipient. This also affects the form of any generated **From** line and the form of envelope addresses used in SMTP commands.

You can also use this attribute when delivering mail to a remote site that is also running **smail** version 3.1. This is useful within a domain that maintains consistent user-forwarding information. This leaves a message in unqualified format until it leaves the domain via a gateway.

local (Boolean)
This implies that attribute **local_xform** is set, but implies that delivery really is the final delivery to a user, file, or program on your local host. This attribute disables generation of a bounce message that results should a message exceed its allowed hop-count.

max_addrs (number)
This attribute sets the maximum number of recipient addresses that can be given in one call to the transport. If this is turned off, then there is no maximum. The default number is one; typically, this attribute either is left at one or turned off.

max_chars (number)
This states the maximum number of characters in the addresses that can be given in one call to this transport. If this is turned off, there is no maximum number. The default number is about one third of the number of characters that can be passed as arguments to a program. When using SMTP transports, this should be turned off unless a remote host is known to be unable to handle a large number of

addresses. For delivery over UUCP to **rmail** ona remote system, this should be in the neighborhood of 200 to 250, to avoid buffer overruns at the remote site. UUCP generally has small buffers to hold argument information.

If **smail** is given an address whose length exceeds this number, then the address will be passed with one call to the transport. Thus, this limit is not strictly enforced.

max_hosts (number)
This states the maximum number of different hosts that can be given in one call to the transport. If this is turned off using the form **-max_hosts**, there is no maximum number. The default number is one and typically this is not changed.

received (Boolean)
If this attribute is set, **smail** inserts a **Received:** field into each message it delivers via this transport. The form of this field is taken from the attribute **received_field** in file **/usr/lib/mail/config**. This attribute is on by default.

return_path (Boolean)
If this attribute is set, **smail** inserts field **Return-Path:** into the header of each message it delivers via this transport. The form of this field is taken from the attribute **return_path_field** in file **/usr/lib/mail/config**. Use this attribute only with a transport that performs final delivery to a local destination.

shadow (string)
This names a second transport through which **smail** also sends the message. This second transport usually performs some task that is unrelated to the actual delivery of the message. For example, you could use a shadow transport to start a program that looked up the sender within a data base and displayed her picture in a window on your workstation. **smail** calls the shadow transport only if the primary transport successfully delivers the message.

strict (Boolean)
If this flag is set, then **smail** attempts to transform mail that does not conform to RFC822 standards. This may be useful for sites that gateway between the UUCP zone and the Internet. In general, it is not a good idea to turn on this attribute, as it changes the contents of headers fields. Turn on this attribute only when you know that some remote hosts understand only mail that conforms to the RFC822 standard.

unix_from_hack (Boolean)
If set, then **smail** inserts the character '>' before any line in the message that begins with the string "From". This is required for local delivery to mailbox files that are in the standard form expected by the System-V program **mailx** and the BSD program **Mail**.

uucp (Boolean)
If set, then **smail** converts outgoing recipient addresses into UUCP-style paths of the form *hosta***!***hostb***!***hostc***!***user*. An exception is that **smail** preserves any use of '%' as an address operator. Thus, **smail** would convert an envelope address of the form **user%hostb@hosta** to **hosta!user%hostb**. This only affects envelope addresses and does *not* affect the body of the message or its header.

inet (Boolean)
If you set this attribute, **smail** converts output-recipient addresses to Internet specifications. This is not the same as the attribute **strict**, because the transformations apply only to the envelope's address, and not to header's. If **inet** is defined, then when **smail** routes a message to a remote system, it generates a "route-addr" address rather than "bang-path" address. Thus, if **smail** is given the address **user%host@gateway** and **gateway** is reached through the path **hosta!hostb!hostc**, then **smail** generates the address **@hostb,@hostc:user%host@gateway**to be sent to the host **@hosta**.

retry_dir (string)
This attribute tells **smail** to use the subdirectory under directory **/usr/lib/mail/retry** for managing host retry intervals for this transport. By default, the directory is named after the transport. However, multiple transports can share a retry directory by using **retry_dir** to force each to use that directory. For example, by default the definition of each TCP/IP SMTP transport uses **retry_dir** to force that transport to use retry directory **smtp**.

remove_header (string)
Tell **smail** to remove the named header field from each message it sends via this transport. This is an expansion string, so header removal can be made dependent upon some condition. If expansion of the string results in an empty string, then no header is removed. You can specify any number of

remove_header attributes for a given transport.

insert_header (string)
append_header (string)
Add the given header field at the beginning (**insert_header**) or end (**append_header**) of the message header for transport. These are expansion strings, so the header (and the existence of the header) can be made to depend on some conditions. If expansion of the string results in an empty string, then **smail** does not add a header. You can specify any number of **insert_header** and **append_header** attributes for a given transport.

The Default Transports

The following describes the transports that are defined in the version of **/usr/lib/mail/transports** that is shipped with COHERENT.

The first transport, **local**, delivers mail to a user on your system:

```
# local - deliver mail to local users
#
# By default, smail will append directly to user mailbox files.
#
local:      driver=appendfile,      # append message to a file
      return_path,            # include a Return-Path: field
      from,                   # supply a From_ envelope line
      local;                  # use local forms for delivery

      file=/usr/spool/mail/${lc:user}, # location of mailbox files
      mode=0600,                       # For BSD: only the user can
                                       # read and write file
      notify_comsat,                   # notify comsat daemon of delivery
      suffix="\1\1\1\10,              # MMDF mailbox format
      prefix="\1\1\1\10,              # MMDF mailbox format
```

The next transport, **pipe**, delivers mail to a shell command:

```
# pipe -deliver mail to shell commands
#
# This is used implicitly when smail encounters addresses which begin with
# a vertical bar character, such as "|/usr/lib/news/recnews talk.bizarre".
# The vertical bar is removed from the address before being given to the
# transport.
pipe: driver=pipe,            # pipe message to another program
      return_path,            # include a Return-Path: field
      from,                   # supply a From_ envelope line
      local;                  # use local forms for delivery

      cmd="/bin/sh -c $user", # send address to the Bourne Shell
      parent_env,             # environment info from parent addr
      pipe_as_user,           # use user-id associated with address
      ignore_status,          # ignore a non-zero exit status
      ignore_write_errors,    # ignore write errors, i.e., broken pipe
      umask=0022,             # umask for child process
      -log_output,            # do not log stdout/stderr
```

The next transport, **file**, delivers mail to a file:

```
# file - deliver mail to files
#
# This is used implicitly when smail encounters addresses which begin with
# a slash or squiggle character, such as "/usr/info/list_messages" or
# perhaps "~/Mail/inbox".
file: driver=appendfile,
      return_path,          # include a Return-Path: field
      from,                 # supply a From_ envelope line
      local;                # use local forms for delivery
```

```
      file=$user,            # file is taken from address
      append_as_user,        # use user-id associated with address
      expand_user,           # expand ~ and $ within address
      mode=0644,             # you may wish to change this
                             # mode, depending upon local
                             # conventions and preferences
      suffix="\1\1\1\10,       # MMDF mailbox format
      prefix="\1\1\1\10,       # MMDF mailbox format
```

The next transport , **uux**, invokes the UUCP command **uux** to deliver messages to a remote site via UUCP:

```
# uux - deliver to the rmail program on a remote UUCP site
#
# HDB UUCP users should comment out the first cmd= line below, and
# uncomment the second.
uux:  driver=pipe,
      uucp,              # use UUCP-style addressing forms
      from,              # supply a From_ envelope line
      max_addrs=5,       # at most 5 addresses per invocation
      max_chars=200;     # at most 200 chars of addresses

      # the -r flag prevents immediate delivery, parentheses around the
      # $user variable prevent special interpretation by uux.
      cmd="/usr/bin/uux - -r -a$sender -g$grade $host!rmail $(($user)$)",
      pipe_as_sender,    # have uucp logs contain caller
      log_output,        # save error output for bounce messages
```

Transport **demand** delivers mail to command **rmail** on a remote system:

```
# demand - deliver to a remote rmail program, polling immediately
#
# HDB UUCP users should comment out the first cmd= line below, and
# uncomment the second.
demand:     driver=pipe,
      uucp,                  # use UUCP-style addressing forms
      from,                  # supply a From_ envelope line
      max_addrs=5,           # at most 5 addresses per invocation
      max_chars=200;         # at most 200 chars of addresses

      cmd="/usr/bin/uux - -a$sender -g$grade $host!rmail $(($user)$)",
      pipe_as_sender,        # have uucp logs contain caller
      log_output,            # save error output for bounce messages
```

The final two transports are local versions of previously defined tranports. What a local transport is, and the advantages it offers, is described above.

Transport **local_uux** is a local version of transport **uux**:

```
local_uux:
      driver=pipe,
      local_xform,       # transfer using local message format
      uucp,              # use uucp-conformant addresses
      from,              # supply a From_ envelope line
      max_addrs=5,       # at most 5 addresses per invocation
      max_chars=200;     # at most 200 chars of addresses

      # the -r flag prevents immediate delivery, parentheses around the
      # $user variable prevent special interpretation by uux.
      cmd="/usr/bin/uux - -r -a$sender -g$grade $host!rmail $(($user)$)",
      pipe_as_sender,    # have uucp logs contain caller
      log_output,        # save error output for bounce messages
```

Finally, **local_demand** is a local form of transport **demand**:

```
local_demand:
      driver=pipe,
      local_xform,         # transfer using local formats
      uucp,                # use uucp-conformant addresses
      from,                # supply a From_ envelope line
      max_addrs=5,         # at most 5 addresses per invocation
      max_chars=200;       # at most 200 chars of addresses

      cmd="/usr/bin/uux - -a$sender -g$grade $host!rmail $(($user)$)",
      pipe_as_sender,      # have uucp logs contain caller
      log_output,          # save error output for bounce messages
```

See Also

Administering COHERENT, config [smail], directors, mail [overview], smail, routers

Notes

For information on how the configuration files **directors**, **routers**, and **transports** relate to each other, see the Lexicon entry for **directors**.

Copyright © 1987, 1988 Ronald S. Karr and Landon Curt Noll. Copyright © 1992 Ronald S. Karr.

For details on the distribution rights and restrictions associated with this software, see file **COPYING**, which is included with the source code to the **smail** system; or type the command: **smail -bc**.

trap — Command

Execute command on receipt of signal
trap [*command***] [***n ...***]**

The command **trap** tells the shell to execute *command* when it receives signal *n*.

You can name more than one signal on the command line for **trap**. Each signal *n* is an integer, as defined in the header file **signal.h**. For information on the traps that COHERENT recognizes and what each one means, see the Lexicon entry for the system call **signal()**. If *n* is zero, the shell executes *command* when it exits.

If you name no *command* on the command line for **trap**, then **trap** resets the trap for signal *n* to its original value. If *command* is a null string (i.e., the string ""), the shell traps signal *n* but does nothing; in effect, this turns off signal *n*.

If you invoke **trap** with no arguments, it prints the signal number and associated command for each signal for which a trap has been set.

The shell executes **trap** directly.

Example

The following example takes two files and outputs only those lines which are the same.

```
# If input only one file-name then simply "cat".
if [ $# = 1 ]; then
      cat $1
      exit 0

# If input two file-names - Ok, else "Usage".
else
      if [ $# != 2 ]; then
            echo "Usage: cmn file1 [file2]"
            exit 1
      fi
fi

# TMP is original name of temporary file (/tmp/temp_(pid)
TMP=/tmp/temp_$$

# Temporary file has to be removed
trap 'rm $TMP; exit 1' 1 2 9
```

```
# Difference between "file1" and "difference between file1 and file2"
# is the common strings "file1" and "file2"
# The strings that are in "file1" and absent in "file2" print in TMP.
diff $1 $2 | sed -n -e "s/^< //p" > $TMP

# The strings that are in "file1" and absent in TMP print in stdout.
diff $1 $TMP | sed -n -e "s/^< //p"

# Remove temporary file
rm $TMP
```

See Also

commands, ksh, sh, signal()

trigraph — C Language

A *trigraph* is a set of three characters that represents one character in the C character set. The set of trigraph sequences was defined in the ANSI Standard to allow users to use the full range of C characters, even if their keyboards do not implement the full C character set. Trigraph sequences are also useful with input devices that reserve one or more members of the C character set for internal use; e.g., the Hazeltine family of terminals, which reserves the tilde '~' as its escape character.

Each trigraph sequence is introduced by two question marks. The third character in the sequence indicates which character is being represented. The following table gives the set of trigraph sequences:

Trigraph Sequence	*Character Represented*
??=	#
??(	[
??/	\
??)	]
??'	^
??<	{
??!	|
??>	}
??-	~

The characters represented are the ones used in the C character set but not included in the ISO 646 character set. ISO 646 describes an invariant sub-set of the ASCII character set.

Trigraph sequences are interpreted even if they occur within a string literal or a character constant. Thus, strings that uses a literal "??" will not work the same as under a non-ANSI implementation of C. For example, the function call

```
printf("Feel lucky, punk??!\n");
```

would print:

```
Feel lucky, punk|
```

To print a pair of questions marks, use the escape sequence '\??'. For example:

```
printf("Feel lucky, punk\??!\n");
```

See Also

cc, C language
ANSI Standard, §5.2.1.1

Notes

By default, the COHERENT C compiler **cc** ignores trigraphs. To invoke interpretation of trigraphs, use the option **-V3GRAPH**.

troff — Command

Extended text-formatting language
troff [*option ...***] [***file ...***]**

The command **troff** is the COHERENT typesetter and text-formatting language. It performs typeset-quality text formatting, suitable for printing on either the Hewlett-Packard LaserJet II or III printers, or on any printer for

which the PostScript language has been implemented.

troff Input

troff processes each given *file*, or the standard input if none is specified, and prints the formatted result on the standard output. The input must consist of text with formatting commands embedded within it.

troff provides a full suite of commands that set line length, page length and page offset, generate vertical and horizontal motions, indentation, fill and adjust output lines, and center text. The great flexibility of **troff** lies in its acceptance of user-defined macros to control almost all higher-level formatting. For example, the formation of paragraphs, header and footer areas, and footnotes must all be implemented by the user via macros.

troff uses a superset of the commands and syntax used by **nroff**, the other COHERENT text-formatter: files prepared for the latter usually can be processed through the former without requiring any changes. **troff** differs from **nroff** in that **nroff** can perform only monospaced formatting, whereas **troff** can handle multiple fonts of type, both monospaced and proportionally spaced. It lets you load font-width tables dynamically, so you can use whatever fonts you have loaded into your printer at a given time. **troff** also lets you move about the page in increments other than sixths of an inch vertically or tenths of an inch horizontally.

troff produces output either in the Hewlett-Packard Printer Control Language (PCL) or PostScript, whichever you prefer. The former can be printed on the Hewlett-Packard LaserJet family of laser printer, and can use any PCL bitmapped "soft font". The latter can be printed on any printer that supports the PostScript language, and can use any font for which you have an Adobe Font Metric description. The default is PCL output; to obtain PostScript, use the **-p** command-line option. See below for information on how to manage downloadable fonts.

Command-line Options

Command-line options may be listed in any order. They are as follows:

-d Debug: print each request before execution. This option is very useful when you are writing and debugging new macros.

-D Display the available fonts. These are all the fonts that have been loaded into **troff** with the **.lf** primitive (described below).

-f *name* Write the temporary file into file *name*.

-i *files* Read from the standard input after reading the given *files*.

-k Keep: do not erase the temporary file.

-l Landscape mode: output is rotated 90 degrees, with default size 11 by 8.5 inches rather than 8.5 by 11 inches.

-m*name* Include the macro file **/usr/lib/tmac.***name* in the input stream.

-n*N* Number the first page of output *N*.

-p Produce output for a PostScript printer rather than for a HP-compatible printer.

-r*aN* Set number register *a* to the value *N*.

-r*abN* Set number register *ab* to value *N*. For obvious reasons, *ab* cannot contain a digit.

-v Return the number of your version.

-x Do not eject to the bottom of the last page when text ends. This option lets you use **troff** interactively, which is especially useful when debugging macros.

If the environmental variable **TROFF** is set when **troff** is invoked, its contents are prefixed to the list of command-line arguments. This allows the user to set commonly used options once in the environment rather than on each **troff** command line.

troff Primitives

As noted earlier, **troff**'s command set is a superset of that used by **nroff**: see the Lexicon entry on **nroff** for information on the commands and escape sequences shared by **troff** and **nroff**. This article describes the primitives that **troff** does *not* share with **nroff**.

Please note that the basic **troff** unit is one-tenth of a point. A printer's point is 1/12 of a pica, which is in turn one-sixth of an inch; therefore, there are 72 points and 720 **troff** units in an inch.

LEXICON

.co *endmark*
Copy input to output file directly, with no processing. If *endmark* argument is present, **troff** copies input until it finds a line containing *endmark* followed by **\n**. If no *endmark* is given, **troff** copies input until it finds a line containing **.co\n**. This directive is useful for embedding PostScript commands in an input file.

.cs *XX N M*
Set font *XX* to use constant character spacing. The width of each character is *N* divided by 36 ems. If *M* is present, it specifies the width of an em; otherwise, *N* assumes the point size em for the given font.

.fd Display the currently available fonts.

.fp *N XX*
Associate font name *XX* with numeric font position *N*. The given *N* should be a number between 1 and 9. Subsequently, the numeric font position can be used in an escape sequence **\f***N* to select the font. (This nomenclature comes from the days when phototypesetters used print wheels that were set in fixed positions on the device.) The **nroff** primitive **.rf** performs a similar task, and is more flexible in its syntax.

.fz *XX N*
Fix the point size of font *XX* at *N*. The point size of the font will not be affected by subsequent **.ps** commands or **\s***N* point size escapes.

.lf *XX file* **[***n***]**
Load font-width table from *file* and use it for font *XX*. If *file* is not found, **troff** looks for **/usr/lib/roff/troff_pcl/fwt/***file* or **/usr/lib/roff/troff_ps/fwt/***file* (depending on whether the **-p** option is used).

The optional third argument sets the default point size of the loaded font to *n*. Note that this argument takes effect only if **troff** is running in **-p** (PostScript) mode.

For example, to load the font-width table for the PCL bitmapped font **cn090rpn.usp** (which sets Century Roman, nine point, portrait mode) and name it font **RS**, use the command:

```
.lf RS cn090rpn.usp
```

To do the same thing under PostScript, use the command:

```
.lf RS Century_R.fwt 9
```

Thereafter, you can reference font **RS** with either **.ft RS** or **\f(RS**.

Note that the second argument to this primitive must name a font-width table generated by the COHERENT command **fwtable**, not the font itself, although both may have the same name.

Please note that **.lf** is unique to the COHERENT implementation of **troff**, and cannot be ported to other implementations.

.ps *N***p** Set point size to *N* points. The default point size is 10 point.

.rb *file* Read input from *file* and copy it to the output without processing. This directive is useful for including files containing PostScript routines in the output.

.ss *N* Set the minimum word spacing to *N* divided by 36 ems.

.vs *N***p** Set the vertical spacing to *N* points. The default vertical spacing for **troff** is 11 points.

Escape Sequences

troff recognizes the following escape sequences, in addition to those recognized by **nroff**:

\| Set a 1/6th-em half-narrow space character.

\^ Set a 1/12th-em half-narrow space character.

\s´*N*˝ Set the point-size escape sequence to *N*. Like the **.ps** primitive, it changes the point size to *N*. The specified *N* may have a leading plus or minus sign to make the new size relative to the current point size.

\X*dd* Output character *dd* where *dd* are two hexadecimal digits. This is useful for forcing **troff** to print characters outside the normal printable range, e.g., those with the high-order bit set. **troff** reserves the following values for its internal use:

<ctrl-space>	0X00	Ignored
<ctrl-A>	0X01	Leader dots, same as "\a"
<ctrl-I>	0X09	Tab, same as "\t"
<ctrl-J>	0X10	Newline

The hexadecimal values to which characters map depend upon the character set that you (or your printer) use. For example, to print the character 'ß' using the Hewlett-Packard Laser-Jet printer and the Pacific Page cartridge, use the escape sequence **\XFB**.

The escape sequence **\X** is unique to the COHERENT implementation of **nroff** and **troff**. Code that uses it will behave differently when ported to other implementations.

Number Registers

The basic unit of measure under **troff** is the decipoint, or one-tenth of a printer's point. A point is one-tenth of a pica, which in turn is one sixth of an inch; therefore, there are 72 points in an inch, or 720 decipoints. All **troff** number registers that hold information about page or type dimensions hold that information in decipoints. For this reason, the decipoint is sometimes called the "machine unit."

The following table shows how other units of measure translate into **troff** machine units:

inch:	1**i** = 720**u**
vertical line space:	1**v** = 110**u**
centimeter:	1**c** = 283**u**
em:	1**m** = 100**u**
en:	1**n** = 50**u**
pica:	1**P** = 120**u**
point:	1**p** = 10**u**

If you are working with PostScript, you must remember to divide the value of a **troff** number register by ten before you pass the value to PostScript, or you will see very strange results on your page — or likelier, no results at all.

Special Characters

troff includes a set of escape sequences for setting special characters. These escape sequences are defined in the files **/usr/lib/roff/troff_*/specials.r**. If you have additional fonts or an extended PostScript cartridge on your printer, you can modify these files to change the current definitions or add new ones.

The following shows the escape sequences currently defined in **specials.r**, and the character each prints:

\(em	—	\(hy	-	\(bu	•	\(sq	[]
\(ru	_	\(14	1/4	\(12	1/2	\(34	3/4
\(fi	fi	\(fl	fl	\(ff	ff	\(Fi	ffi
\(Fl	ffl	\(de	°	\(dg	†	\(fm	'
\(ct	¢	\(rg	®	\(co	©	\(tm	™
\(pl	+	\(mi	–	\(eq	=	\(**	*
\(sc	§	\(aa	´	\(ga	`	\(ul	_
\(sl	/	\(*a	α	\(*b	β	\(*g	γ
\(*d	δ	\(*e	ε	\(*z	ζ	\(*y	η
\(*h	θ	\(*i	ι	\(*k	κ	\(*l	λ
\(*m	μ	\(*n	ν	\(*c	ξ	\(*o	ο
\(*p	π	\(*r	ρ	\(*s	σ	\(ts	ς
\(*t	τ	\(*u	υ	\(*f	φ	\(*x	χ
\(*q	ψ	\(*w	ω	\(*A	Α	\(*B	Β
\(*G	Γ	\(*D	Δ	\(*E	Ε	\(*Z	Ζ
\(*Y	Η	\(*H	Θ	\(*I	Ι	\(*K	Κ
\(*L	Λ	\(*M	Μ	\(*N	Ν	\(*C	Ξ
\(*O	Ο	\(*P	Π	\(*R	Ρ	\(*S	Σ
\(*T	Τ	\(*U	Υ	\(*F	Φ	\(*X	Χ
\(*Q	Ψ	\(*W	Ω	\(sr	√	\(rn	‾
\(>=	≥	\(<=	≤	\(==	≡	\(~=	≈
\(ap	~	\(!=	≠	\(->	→	\(<-	←
\(ua	↑	\(da	↓	\(mu	×	\(di	/
\(+-	±	\(cu	∪	\(ca	∩	\(sb	⊂
\(sp	⊃	\(ib	⊆	\(ip	⊇	\(in	∞
\(pd	∂	\(gr	∇	\(no	¬	\(is	∫

LEXICON

\(pt	∝	\(es	∅	\(mo	∈	\(br	│
\(dd	‡	\(rh	☞	\(lh	<-	\(or	|
\(ci	O	\(lt	⎛	\(lb	⎝	\(rt	⎠
\(rb	⎭	\(lk	⎨	\(rk	⎬	\(bv	⎪
\(lf	⎣	\(rf	⎦	\(lc	⎡	\(rc	⎤

Printer Configuration

troff reads several files in directory **/usr/lib/roff/troff_pcl** (when generating PCL output) or **/usr/lib/roff/troff_ps** (when generating PostScript) to find printer-specific information. It reads special character definitions from file **specials.r**. It reads font loading requests from file **fonts.r**. It copies file **.pre** at the beginning of the output. It copies file **.post** at the end of the output. In landscape mode, **troff** looks for files **.pre_land** and **.post_land** instead. You can change these files as desired to include printer-specific commands in **troff** output.

Managing Fonts

As noted above, **troff** produces output in either of two page-description languages: the Hewlett-Packard Printer Control Language (PCL), which is the "native language" of Hewlett-Packard's LaserJet printers; or PostScript. The COHERENT system also comes with tools that lets you process fonts, so that you can use with **troff** either downloadable soft fonts or the fonts that are on board your printer.

The following two sections describe how to manage fonts under PCL and under PostScript. You should refer to the section that is appropriate to your type of printer.

PCL Fonts

Before **troff** can use a font, it must know the following information:

- What the width of every character of the font is, and
- How it can tell the printer to print that font.

Both pieces of information are stored in a file called a *font-width table*. Before **troff** can use a font, it must read the font-width table for that font.

To load a font-width table into **troff**, use the primitive **.lf**. Its syntax is as follows:

.lf *XX file*

XX gives the name by which you will call the font in your **troff** program. *file* is the font-width table for this font. If *file* is not a full path name, **troff** looks for it in directory **/usr/lib/roff/troff_pcl/fwt**.

COHERENT comes with font-width tables for a number of commonly used fonts. The following tables are for the fonts built into the Hewlett-Packard LaserJet III:

Table	*Description*
CGTimes_B.fwt	Times Bold, scalable, rotatable
CGTimes_BI.fwt	Times Bold Italic, scalable, rotatable
CGTimes_I.fwt	Times Italic, scalable, rotatable
CGTimes_R.fwt	Times Roman, scalable, rotatable
Cour10_B.fwt	Courier Bold, ten point, portrait
Cour10_I.fwt	Courier Italic, ten point, portrait
Cour10_R.fwt	Courier Roman, ten point, portrait
Cour12L_B.fwt	Courier Bold, 12 point, landscape
Cour12L_R.fwt	Courier Roman, 12 point, landscape
Cour12_B.fwt	Courier Bold, 12 point, portrait
Cour12_I.fwt	Courier Italic, 12 point, portrait
Cour12_R.fwt	Courier Roman, 12 point, portrait
LinepL_R.fwt	Line Printer, 8.5 point, landscape
Linep_R.fwt	Line Printer, 8.5 point, portrait
Univers_B.fwt	Univers Bold, scalable, rotatable
Univers_BI.fwt	Univers Bold Italic, scalable, rotatable
Univers_I.fwt	Univers Italic, scalable, rotatable
Univers_R.fwt	Univers Roman, scalable, rotatable

Note that the scalable Hewlett-Packard fonts are set by default at 250 points in size — that is, about 3.5 inches. Because you cannot scale PCL fonts when you load them, you must use the **.ps** primitive to size the font.

The following **troff** program demonstrates scalable fonts on the Hewlett-Packard LaserJet III:

```
.lf TR CGTimes_R.fwt
.lf TB CGTimes_B.fwt
.lf TI CGTimes_I.fwt
.lf UR Univers_R.fwt
.lf UB Univers_B.fwt
.lf UI Univers_I.fwt
.vs 14p
.ps 12p
\f(TRThis is Times Roman, 12 point.
.sp
\f(TBThis is Times Bold, 12 point.
.sp
\f(TIThis is Times Italic, 12 point.
.vs 26p
.ps 24p
\f(URThis is Univers Roman, 24 point.
.sp
\f(UBThis is Univers Bold, 24 point.
.sp
\f(UIThis is Univers Italic, 24 point.
.br
```

Note that this program does not run correctly if downloaded to a LaserJet II, or to any printer that is running PostScript.

The COHERENT command **fwtable** lets you build new font-width tables. It can build tables for PCL bit-mapped soft fonts, as well as for fonts that are built into the LaserJet III.

To manipulate PCL bit-mapped soft fonts, do the following:

- Use the command **fwtable** to build a font-width table from the font. The input to **fwtable** should be the soft font itself; and the output of **fwtable** should be redirected into an appropriately named file. See the lists of tables given above for an idea of how to name your font-width table.
- Move the newly created font-width table into directory **/usr/lib/roff/troff_pcl/fwt**.
- Move the font itself into directory **/usr/lib/roff/troff_pcl/fonts**. You may need to create this directory if this is the first time you are using soft fonts.
- Include the instruction **.lf** in your **troff** file to load the font-width table and name the font, as shown above. If you use the same fonts repeatedly, you may wish to put the **.lf** primitives into a separate file that you always include on your **troff** command line via the environmental variable **TROFF**.
- Before you print your document, load the soft font into your printer. If you are using the **hp** spooler to spool files to your printer, use the command **hpr -f**. If you are using the MLP spooler, then you must pre-process the font with the command **pclfont**, then spool the processed font to device **hpraw**. Both commands are described in detail in their Lexicon entries. Briefly, to load font **tr100bpn.usp** into your printer, use the command

  ```
  hpr -f /usr/lib/roff/troff_pcl/fonts/tr100bpn.usp
  ```

  or the command:

  ```
  pclfont /usr/lib/roff/troff_pcl/fonts/tr100bpn.usp | lp -d hpraw
  ```

  These commands also let you specify what “slot” to put the font; you can use this to help manage fonts in your printer. By placing the frequently used fonts in the lower slots, you can then load the less-frequently used fonts into the upper slots, and overwrite just those fonts when you change fonts for another printing job. You must do such font management by hand — COHERENT does not include a utility to do it for you.

  You may wish to write the font-loading commands into a script that you execute before you print a job. You must reload fonts every time you power up your printer or clear its memory.

To build a font-width table for a font built into your LaserJet III, do the following:

- Each font on your printer is described with a **.tfm** file, which comes on a disk with your printer. (If you did not receive such a disk, check with the dealer from which you purchased your printer, or write to Hewlett-Packard.) Use the COHERENT command **doscp** to copy the **.tfm** file for the font that interests you from the disk.

LEXICON

- Use the command **fwtable -t** to build the font-width table. Its input should be the **.tfm** file that you just uploaded. Redirect its output into an appropriate named file.
- Move the newly created font-width table into directory **/usr/lib/roff/troff_pcl/fwt**.
- Note that because the font is build into your printer, you do not need to download anything before you can use the font. When **troff** reads the font-width table, it will know how to invoke the font on your printer.

PostScript Fonts

Before **troff** can use a font, it must know the following information:

- What the width of every character of the font is, and
- How it can tell the printer to print that font.

Both pieces of information are stored in a file called a *font-width table*. Before **troff** can use a font, it must read the font-width table for that font.

To load a font into **troff**, use the primitive **.lf**. Its syntax is as follows:

.lf *XX file* **[***n***]**

XX gives the name by which you will call the font in your **troff** program. *file* is the font-width table for this font. If *file* is not a full path name, **troff** looks for it either in directory **/usr/lib/roff/troff_ps/fwt**.

The optional argument **n** lets you size the font. This applies only to PostScript scalable fonts. All fonts that are loaded with this option are *not* affected by the **.ps** primitive.

For example, the instruction

```
.lf HR      HelvNar_R.fwt  12
```

loads a font for PostScript output. The font is named **HR**. The font-width table is read from file **/usr/lib/roff/troff_ps/HelvNar_R.fwt**, which defines the font Helvetica Narrow Roman. Finally, it sizes the font to 12 points. Hereafter, the instructions **.ft HR** or **\f(HR** invoke this font.

COHERENT comes with font-width tables for a number of commonly used fonts. The following tables are for PostScript fonts. LaserJet III, and are kept in directory **/usr/lib/roff/troff_pcl/fwt**. All are, of course, scalable and rotatable:

Table	*Description*
Avant_B.fwt	Avant-Garde Roman (Gothic Book)
Avant_BI.fwt	Avant-Garde Bold Italic
Avant_I.fwt	Avant-Garde Italic
Avant_R.fwt	Avant-Garde Roman
Bookman_B.fwt	Bookman Bold
Bookman_BI.fwt	Bookman Bold Italic
Bookman_I.fwt	Bookman Italic
Bookman_R.fwt	Bookman Roman
Century_B.fwt	Century Bold
Century_BI.fwt	Century Bold Italic
Century_I.fwt	Century Italic
Century_R.fwt	Century Roman
Chancery_I.fwt	Zapf Chancery Italic
Courier_B.fwt	Courier Bold
Courier_BI.fwt	Courier Bold Italic
Courier_I.fwt	Courier Italic
Courier_R.fwt	Courier Roman
Dingbats.fwt	Zapf Dingbats
HelvNar_B.fwt	Helvetica Narrow Bold
HelvNar_BI.fwt	Helvetica Narrow Bold Italic
HelvNar_I.fwt	Helvetica Narrow Italic
HelvNar_R.fwt	Helvetica Narrow Roman
Helv_B.fwt	Helvetica Bold
Helv_BI.fwt	Helvetica Bold Italic
Helv_I.fwt	Helvetica Italic
Helv_R.fwt	Helvetica Narrow

Pala_B.fwt . . . . . Zapf Calligraphic Bold (Palatino)
Pala_BI.fwt . . . . . Zapf Calligraphic Bold Italic
Pala_I.fwt . . . . . Zapf Calligraphic Italic
Pala_R.fwt . . . . . Zapf Calligraphic Roman
Symbol.fwt . . . . . Symbols
Times_B.fwt. . . . . Times Bold
Times_BI.fwt . . . . . Times Bold Italic
Times_I.fwt . . . . . Times Italic
Times_R.fwt. . . . . Times Roman

Note that these tables are designed for the fonts used on the Pacific Page implementation of the PostScript language. They may not work correctly with genuine Adobe fonts.

The following gives an example program to demonstrate the PostScript fonts:

```
.lf HR      HelvNar_R.fwt  12
.lf HC      Avant_B.fwt    24
.lf DB      Dingbats.fwt    9
.vs 14
.sp
\f(HRThis is 12-point Helvetic Narrow Roman
.vs 26
.sp
\f(HCThis is 24-point Avant-Garde
.vs 11
.sp
\fRA row of dingbats:  \f(DBa row of dingbats
```

This program will not work unless you format using the **-p** option to **troff**, and print it on a PostScript printer. Please note that because PostScript is a portable language, you can print the PostScript output of **troff** on any printer that implements PostScript, not just the Hewlett-Packard LaserJet.

COHERENT comes with tools with which you can "cook" fonts so that you can use with with **troff**, whether the fonts are downloadable soft fonts or on board a cartridge. To cook fonts that are on-board a cartridge in your printer, do the following:

- First, the PostScript cartridge should come with a set of files that give font-width information. These have the suffix **.afm**; there should be one file for each font in your cartridge. If you did not receive such a cartridge, contact the dealer from which you purchased the cartridge, or contact the cartridge's manufacturer. Use the command **doscp** to copy the **.afm** files from the disk onto your COHERENT system.
- Use the command **fwtable -p** to cook each **.afm** file into **troff**'s font-width table format. Each font-width table that you create should have the suffix **.fwt**, and should be named so that it appropriate describes the font. See the above table of font-width tables for examples.

  Move the newly created font-width tables into directory **/usr/lib/roff/troff_ps/fwt**.

  Thereafter, when you write a **troff** program, use the **.lf** primitive to load the font-width table. You may wish to create a file called **fonts.r** that routinely loads all of the font-width tables that you use routinely. You do *not* need to load fonts into your printer; the font-width table includes the information needed so that **troff** can invoke them from your cartridge.

COHERENT comes with tools to help you manage download soft fonts under PostScript. Note that the fonts must be in the Adobe Font Metric (AFM) format. To manage downloadable AFM fonts, do the following:

- A downloadable AFM font comes in three files: a file of information about the font, which has the suffix **.inf**; a file that contains the font-width table, which has the suffix **.afm**; and a file that contains the font itself, which has the suffix **.pfb**. You can ignore the **.inf** file; it is not used in this process. You should use the COHERENT command **doscp -b** to copy the **.pfb** from the floppy disk; and use the command **doscp -a** to copy the **.afm** file from floppy disk. (The options **-b** and **-a** stand, respectively, for binary and ASCII modes.)
- Use the command

  `fwtable -p` *fontname*`.afm` *fontname*`.fwt`

  to generate the font-width table from the **.afm** file. Note that the font-width table should have the suffix **.fwt**. By convention, you should give the font-width table the same name as the font, to help you remember which table goes with which font; this, however, is not required. For example, to create the font-width table for the Adobe font Avant Garde bold, use the following command:

```
fwtable -p avgb____.afm avgb____.fwt
```

- Move the newly created font-width table into directory **/usr/lib/roff/troff_ps/fwt**.
- Next, use the command **PSfont** to "cook" the **.pfb** file into a form that can be downloaded to your printer. Note that a font can be cooked into either of two forms. The first form permits the font to stay resident in your printer, so that you can use it to print an indefinite number of documents. The second form does not permit the font to stay resident in your printer, but it does permit you to include the font directly within your **troff** output. The first form is the default output of **PSfont**; to create the second form, invoke **PSfont** with its option **-s**. For example, to cook the font Avant Garde bold into the first output format, use the command:

  ```
  PSfont avgb____.pfb avgb____.ps1
  ```

  To it into the second form, use the command:

  ```
  PSfont -s avgb____.pfb avgb____.ps2
  ```

  Note that the suffix **.ps1** indicates the first (stay-resident) form of the font, whereas the suffix **.ps2** indicates the second (includable) form of the font. These suffixes are simply conventions, and are not required.
- Move the newly created fonts into directory **/usr/lib/roff/troff_ps/ps**. Note that you may need to create this directory when you first begin to process fonts.
- When you create a **troff** program, use the primitive **.lf** to include the font-width table for this font and size the font, as described above.
- If you have processed the fonts into the first (stay-resident) form, you must load them into your printer before you can print any documents. To download the font, use either the command **hpr -B** or the command **lp -d***printer* (where *printer* names the printer to which the font is being downloaded). For example, to download the Avant Garde bold font to printer **hpraw**, use the command:

  ```
  lp -dhpraw /usr/lib/roff/troff_ps/ps/avgb____.ps1
  ```

  (For more information on the command **lp**, see its entry in the Lexicon, or see the entry for **printer**.) You may wish to create a script to download the fonts that you use commonly. Note that you must reload the fonts into your printer every time you either power up the printer or clear out its memory. Note, too, that downloading and processing stay-resident fonts may take several minutes, depending upon your printer's make.
- To use the "includable" form of a font, use the **troff** primitive **.rb** to load it into the **troff**. For example, to include Avant Garde bold directly within your **troff** output, include the following statement in your **troff** source:

  ```
  .rb /usr/lib/roff/troff_ps/ps/avgb____.ps2
  ```

  If you use some downloadable fonts commonly, you may wish to include a set of **.rb** statements for the fonts in file **fonts.r**. Note that files that include downloadable fonts will be *much* larger than those that do not use them.

Files

/tmp/rof* — Temporary files
/usr/lib/tmac.* — Standard macro packages
/usr/lib/roff/troff_pcl/ — Support files directory for PCL
/usr/lib/roff/troff_ps/ — Support files directory for PostScript
/usr/lib/roff/troff_*/.pre — Output prefix
/usr/lib/roff/troff_*/.pre_land — Output prefix, landscape mode
/usr/lib/roff/troff_*/.post — Output suffix
/usr/lib/roff/troff_*/.post_land — Output suffix, landscape mode
/usr/lib/roff/troff_*/fonts.r — Font definitions
/usr/lib/roff/troff_*/fwt/ — Directory for font width tables
/usr/lib/roff/troff_*/specials.r — Special character definitions

See Also

col, commands, deroff, fwtable, hpr, lp, man, ms, nroff, printer, PSfont
nroff, The Text-Formatting Language, tutorial

Adobe Systems Incorporated: *PostScript Language Reference Manual.* Reading, Mass.: Addison-Wesley Publishing Company, Inc., 1988.

Adobe Systems Incorporated: *PostScript Language Tutorial and Cookbook.* Reading, Mass.: Addison-Wesley Publishing Company, Inc., 1988.

Emerson, S.L., Paulsell, K.: *troff Typesetting for Unix Systems.* Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1987 (ISBN 0-13-930959-4).

Lawson, A.: *Printing Types: An Introduction.* Boston: Beacon Press, 1971.

Lawson, A.: *Anatomy of a Typeface.* Boston: David R. Godine, Publisher, 1990.

Diagnostics

For a list of the error messages that **troff** can produce, see the Lexicon entry for **nroff**.

Notes

Like **nroff**, **troff** should be used with the macro packages **ms**, which is found in the file **/usr/lib/tmac.s**, and **man**, which is found in the file **/usr/lib/tmac.an**.

troff output, unlike that of **nroff**, cannot be processed through a terminal driver. If you redirect the output of **troff** to a terminal, all you will see is the literal program it outputs.

Laser printers cannot print on an area near each edge of the output page. Output sent to the unprintable area will disappear. On some printers, the *logical page* does not correspond to the *physical page*, so printed **troff** output may be offset from the specified position on the physical page.

true — Command

Unconditional success
true

true does nothing, successfully. It always returns zero (i.e., true).

true is useful in shell scripts when you want to execute a condition indefinitely. For example, the following example

```
while true; do
      date
done
```

prints the current date and time on your screen forever (or at least until interrupted by typing **<ctrl-C>**).

See Also

commands, false, ksh, sh

Notes

Under the Korn shell, **true** is an alias for the partial-comment **:**.

trustme — System Administration

List of trusted users
/etc/trustme

The file **/etc/trustme** names users who are "trusted" — that is, who are permitted to log into the system even though the file **/etc/nologin** has been created to stop users from logging in.

See Also

Administering COHERENT, login, nologin

tsort — Command

Topological sort
tsort [*file***]**

tsort performs a topological sort of a set of input items. The input *file* (or the standard input, if no *file* is given) specifies an ordering on pairs of items. It consists of pairs of items separated by blanks, tabs or newlines. If a pair contains the same item twice, it simply indicates that the item is in the input set. Otherwise, the pair indicates that the first item precedes the second in the ordering.

tsort prints a sorted list of the input items on the standard output.

See Also

commands, sort

Diagnostics

tsort prints an error message on the standard error if its input contains an odd number of items or if the specified ordering includes a cycle.

ttt — Command

Play 3-D tic-tac-toe
/usr/games/ttt

The COHERENT game **ttt** plays three-dimensional tic-tac-toe. Each playing board is four-by-four, and four are stacked on top of each other. You play against the computer; each player selects to occupy one "square" on one of the boards. The first player to get four four squares in a row, in any direction, wins.

See Also

commands

tty — Command

Print the user's terminal name
tty

tty prints the name of the character-special file that manages your terminal.

Diagnostics

tty prints the message "Not a tty." if the user is not associated with any controlling terminal.

See Also

commands, who

tty.h — Header File

Define flags used with tty processing
#include <sys/tty.h>

tty.h defines manifest constants that are used by the routines that handle ttys.

See Also

header files, tty

ttyname() — General Function (libc)

Identify a terminal
#include <unistd.h>
char *ttyname(*fd***)**
int *fd***;**

Given a file descriptor *fd* attached to a terminal, **ttyname()** returns the complete pathname of the special file (normally found in the directory **/dev**).

Files

/dev/* — Terminal special files
/etc/ttys — Login terminals

See Also

ioctl(), isatty(), libc, tty(), ttyslot(), unistd.h
POSIX Standard, §4.7.2

Diagnostics

ttyname() returns NULL if it cannot find a special file corresponding to *fd*.

Notes

The string returned by **ttyname()** is kept in a static area, and is overwritten by each subsequent call.

ttys — System Administration

Describe terminal ports
/etc/ttys

File **/etc/ttys** describes the terminals in the COHERENT system. The process **init** reads this file when it brings up the system in multi-user mode.

/etc/ttys contains one line for each terminal. Each line consists of the following four fields:

1. The first field is one character long, and indicates if the device is enabled for logins: '0' indicates that the device is not enabled, and '1' (one) indicates that logins are enabled for the device.

2. The second field is one character long, and indicates whether the device is local (i.e., a terminal) or remote (i.e., a modem): 'r' indicates remote, and 'l' (lower-case **L**) indicates local.

   If the port is named in file **/etc/dialups**, then the command **login** checks the file **/etc/d_passwd** to see if the program the user is invoking is protected by a password. If so, it prompts the user for that additional password before allowing her to log in. For details, see the Lexicon entries for **login**, **dialups**, and **d_passwd**.

3. The third field is one character long, and sets the baud rate for the device. Note that a device can have either a fixed baud rate, or a variable baud rate. The following table gives the codes for fixed baud rates:

C	110
G	300
I	1200
L	2400
N	4800
P	9600
Q	19200
S	38400

   The common variable-speed codes terminal types are as follows:

0	300, 1200, 150, 110
3	2400, 1200, 300

   When a user dials into a variable-speed line, a message is sent to the terminal using the first speed listed. If the message is unintelligible, the user hits the **<break>** key and the system tries the next speed; and so on, until the correct speed is selected.

4. The fourth field names the port that this device is plugged into. The following table names the ports that COHERENT recognizes:

console	The console device
color*N*	Virtual console device *N*, color console
mono*N*	Virtual console device *N*, monochrome console
com*N***l**	Serial port **com***N*, local device
com*N***r**	Serial port **com***N*, remote device
com*N***fl**	Serial port **com***N*, local device, flow control
com*N***fr**	Serial port **com***N*, remote device, flow control
com*N***pl**	Serial port **com***N*, local polled device
com*N***pr**	Serial port **com***N*, remote polled device

   Note that if field 2 (described above) says that this is a local device, then you must use a port descriptor that ends in 'l'; likewise, if field 2 states that this is a remote device, the port descriptor must end in 'r'. Doing otherwise will result in trouble. See Lexicon entry **asy** for details. Note also that you must use a device with hardware flow control (i.e., a device whose suffix includes the letter 'f') if you wish to use a high-speed modem (e.g., 14.4*bis*).

Do not leave trailing spaces at the end of an entry in **/etc/ttys**. Leaving blanks at the end of a line usually results in errors that state that a device could not be found.

After you have edited **/etc/ttys**, the following command forces COHERENT to re-read the file and use the new descriptions:

```
kill quit 1
```

Examples

Consider the following **ttys** entry:

```
1lPconsole
```

Field 1 is the first character. Here it is set to '1' (one), which indicates that the device is enabled for logins. Field 2 is the second character. Here it is set to 'l' (lower-case **L**), which indicates that this is a local device. Field 3 is the third character. Here, it is set to 'P', which indicates that the device operates at the fixed baud rate of 9600 baud. This field is ignored by the console device driver since the console is not a serial device. Finally, field 4 is the remainder of the line. Here, it indicates that the device in question is the console.

Now, consider another example:

```
1r3com3r
```

Field 1 is the first character. Here it is set to '1' (one), which indicates that the device is enabled for logins. Field 2 is the second character. Here it is set to 'r', which indicates that this is a remote device, i.e., a modem. Field 3 is the third character. Here, it is set to '3', which indicates that the device operates at variable baud rates of 2400, 1200, and 300. By hitting the **<break>** key on the terminal, the user can select from among those three baud rates, in that order. Finally, field 4 is the remainder of the line. Here, it indicates that the device in question is plugged into port **com3**, and is accessed via special file **/dev/com3r**.

Files

/etc/ttys

See Also

Administering COHERENT, asy, d_passwd, dialups, getty, init, login, stty, terminal, tty

Notes

If you wish to enable logins on a COM port on which you will also be dialing out, you must edit file **/etc/ttys** and add a line for the raw device. For example, if you have a modem plugged into COM1 and you wish to dial out on that port, you must have an entry for both **com1l** and **com1r**. Note that the entry for **com1r** *must* precede the entry for **com1l**. If you do not do this, the commands **cu** and **uucico** cannot disable **com1r** before they dial out on **com1l**.

cu also requires that the device **/dev/console** appear last in file **/etc/ttys**. If this is not so, **cu** refuses to disable the enabled port or dial out.

ttyslot() — General Function (libc)

Return a terminal's line number
int ttyslot()

ttyslot() returns the number of the line in the file **/etc/ttys** that describes the controlling terminal (see **ttys**).

Files

/dev/* — Terminal special files
/etc/ttys — Login terminals

See Also

libc

Diagnostics

ttyslot() returns zero if an error occurs.

ttystat — Command

Get terminal status
/etc/ttystat [-d] *port*

ttystat checks the status of the specified asynchronous *port* in directory **/dev**. It normally just returns an exit status that indicates the status of the *port*. The option **-d** tells **ttystat** to print the status of the *port* on the standard output.

Example

The following example prints the status of port **/dev/com2r**:

```
/etc/ttystat -d com2
```

If **/dev/com2r** is enabled, **ttystat** prints:

```
com2r is enabled
```

ttystat finds the port status from the **/etc/ttys** file.

Files

/etc/ttys — Terminal characteristics file

See Also

commands, disable, enable, ttys

Diagnostics

ttystat returns one if the *port* is enabled, or zero if the *port* is disabled. It returns -2 if an error occurs.

ttytype — Command

Select a default terminal type for a port
ttytype

The command **ttytype** selects a default terminal type for a given port.

The default terminal types are recorded in file **/etc/ttytype**. You must edit this file to ensure that the default terminal types are described correctly. The following gives a example version of **/etc/ttytype**:

```
ansipc    console    The COHERENT console
adm3a     com1l      The old Kaypro II
vt100     com2r      Remote logins
```

The first string gives the type of terminal. This string must name a terminal that is recognized by **termcap** and **terminfo** The second string gives the device with which this terminal type is linked. The **console** device should always be linked to terminal type **ansipc**. Other devices can be linked to the type of terminal most often used on them; on the above example, the user has a Kaypro II that is wired into his COHERENT system via a local serial port. **ttytype** ignores all strings after the first two in each line, so you can add comments to each entry, as in the above example.

You can use **ttytype** to set a terminal type automatically at login time. To do so, edit the file **/etc/profile** and replace the line

```
export TERM=ansipc
```

with the command:

```
export TERM=`/usr/bin/ttytype`
```

Files

/etc/ttytype — File of default terminal types

See Also

commands, termcap, terminfo

type checking — Definition

Every expression has a *type*, such as **int**, **char**, or **double**. C is not strongly typed, which means that it allows different types to be mixed relatively freely, and be changed (or **cast**) from one type to another.

COHERENT checks types more strictly than the C standard implies. COHERENT's type checking can be enabled or disabled in degrees, using **-VSTRICT** and other "variant" options with the **cc** command.

See Also

cc, Programming COHERENT, type promotion

type promotion — Definition

In arithmetic expressions, COHERENT promotes one signed type to another signed type by sign extension, and promotes one unsigned type to another unsigned type by zero padding. For example, **char** promotes to **int** by sign extension, whereas **unsigned char** promotes to **unsigned int** by zero padding.

See Also

data formats, Programming COHERENT

typedef — C Keyword

Define a new data type

typedef is a C facility that lets you define new data types. Such definitions are always made in terms of existing data types; for example,

```
typedef long time_t;
```

establishes the data type **time_t**, and defines it to be equivalent to a **long**. By convention, programmer-defined data types are written in capital letters.

Judicious use of the **typedef** facility can make programs easier to maintain, and improve their portability.

See Also

C keyword, manifest constants, portability, storage class
ANSI Standard, §6.5.6

types.h — Header File

Define system-specific data types
#include <sys/types.h>

The header file **types.h** defines a number of data types that are used throughout the COHERENT system.

See Also

header_fi
POSIX Standard, §2.5

typeset — Command

Set/list variables and their attributes
typeset
typeset [+-]fr
typeset [irx] *variable*=*value*

The command **typeset** is built into the Korn shell **ksh**. It sets or lists all variables and their attributes.

When called with an argument of the form *variable*=*value*, it sets variable *variable* to *value.* The following options modify *variable* or *value*:

- **i** Store *value* as an integer
- **r** Make *variable read-only*
- **x** Export *variable* to the environment

When called without an argument, **typeset** lists all variables and their attributes. When called with one of the following options, it lists the variables of the appropriate type. When prefixed with a hyphen '-', it prints the variable plus its value; when prefixed with a plus sign '+', it prints the variable alone:

- **f** List functions instead of variables
- **r** List read-only variables

See Also

commands, ksh

typo — Command

Detect possible typographical and spelling errors
typo [-nrs][*file ...***]**

typo proofreads an English-language document for typographical errors. It conducts a statistical test of letter digrams and trigrams in each input word against digram and trigram frequencies throughout the entire document. From this test, **typo** computes an index of peculiarity for each word in the document. A high index indicates a word less like other words in the document than does a low index. Built-in frequency tables ensure reasonable results even for relatively short documents.

typo reads each input *file* (or the standard input if none), and removes punctuation and non-alphabetic characters to produce a list of the words in the document. To reduce the volume of the output, **typo** compares each word against a small dictionary of technical words and discards it if found. The output consists of a list of unique non-dictionary words with associated index of peculiarity, most peculiar first. An index higher than ten indicates that the word almost certainly occurs only once in the document.

typo recognizes the following arguments:

-n Inhibit use of the built-in English digram and trigram statistics, and inhibit dictionary screening of words. More words will be output and the indices of peculiarity will be less useful for short documents.

-r Inhibit the default stripping of **nroff** escape sequences. Normally, **typo** strips lines beginning with '.' and removes the **nroff** escape sequences '\'.

-s Produce output files **digrams** and **trigrams** that contain, respectively, the digram and trigram frequency statistics for the given document. No indices of peculiarity are calculated or printed. If desired, these files may be installed in directory **/usr/dict**.

Files

/tmp/typo* — Intermediate files
/usr/dict/dict — Limited dictionary
/usr/dict/digrams — Digram frequency statistics
/usr/dict/trigrams — Trigram frequency statistics

See Also

commands, nroff, sort, spell

tzset() — Time Function (libc)

Set the local time zone
#include <time.h>
#include <sys/types.h>
void tzset()
extern long timezone; char *tzname[2][16];

tzset() is one of the suite of COHERENT functions that control and display the system's time. It searches for the environmental parameter **TIMEZONE**, which gives information on the local time zone. For more information on **TIMEZONE**, see its Lexicon entry.

If **TIMEZONE** is set, **tzset()** initializes the external variables **timezone** and **tzname**. **timezone** contains the number of seconds to be subtracted from GMT to obtain local standard time. **tzname[0]** and **tzname[1]** are character arrays that hold, respectively, the names of the local standard time zone and the local daylight-saving time zone. If **TIMEZONE** is not set, **timezone** defaults to zero, **tzname[0]** to **GMT**, and **tzname[1]** to the empty string.

See Also

date, ftime(), libc, localization, time [overview], TIMEZONE

Notes

tzset() used to be named **settz()**. It has been renamed to conform to published standards.