n.out.h — Header File

Define n.out file structure
#include <n.out.h>

n.out.h defines the **n.out** file structure. It is the same as the standard COHERENT form **l.out**, except that it uses 32-bit addressing. This file structure is used internally in COHERENT, but is not available under the COHERENT C compiler or assembler.

See Also

coff.h header files, l.out.h

name space — C Language

C name-space rules
The term
name space
refers to the “list” where the translator records an identifier.
Each name space holds a different set of identifiers.
If two identifiers are spelled exactly the same and appear within
the same scope but are not in the same name space,
they are *not* considered to be identical.

The five varieties of name space, as follows:

Macro Names
Macro names introduced with **#define** are special. Because macro replacement happens before the program text is scanned for the other classes of names, macro names exist in a global name space that pays no heed to the rules below. See the description of name-space pollution, below, for more on this.

Label Names
The translator treats every identifier followed by a colon ‘:’ or that follows a **goto** statement as a label.

Tags A tag is the name that follows the keywords **struct**, **union**, or **enum**. It names the type of object so declared.

Members
A member names a field within a structure or a **union**. A member can be accessed via the operators ‘.’ or ‘->’. Each structure or **union** type has a separate name space for its members.

Ordinary identifiers
These name ordinary functions and variables. For example, the expression

```
int example;
```

declares the ordinary identifier **example** to name an object of type **int**.

Name-Space Pollution

The ANSI Standard and the POSIX Standard recognize special problems that relate to the above classes of name space and to the names supplied to the user by the translator or the **#include** mechanism. They provide special rules that govern what names a program and an implementation can define.

Although the above rules are good at resolving conflict, in the context of a large programming project (which the standard C library is, effectively) they are not always sufficient. First, there is the possibility that definitions in library header files may conflict with each other, or with user definitions. Second, an internal definition in the standard library may conflict with a user definition that happens to have the same name.

The ANSI Standard defines rules that set aside some names for the implementation. The implementation can use only these names, and user applications cannot use them. When implementations and applications both obey these rules, a user program cannot conflict with a definition in a system header file. The rules are as follows:

- Any name that begins with an underscore followed by a capital letter or underscore is reserved for use by the implementation. Applications should not use any symbols of this form except to define feature-test macros (e.g., **_POSIX_SOURCE**, see below).
- Any name that begins with an underscore followed by a lower-case letter is reserved for use by the application if the name is internal (such as a static symbol or a tag- or member-name). Macro names of this form are forbidden, because they do not obey the other name-space rules above: a user-level macro definition could cause a conflict with a private structure-member defined in a system header.
- C++ reserves for the implementation all names that contain two underscores.
- The Standard forbids external identifiers (i.e., non-static functions and variables) that match any of the function or variable defined in the C standard.
- If a program **#include**s a standard library header file, it cannot use a macro definition that matches the name of any function or variable defined in any standard library header.

These rules are supplemented with rules that govern the use of names that are defined in any library header described in the ANSI Standard or the POSIX Standard. The following gives the rules that apply to individual header files:

<errno.h>
The implementation can define extra macros that begin with the letter 'E'.

<signal.h>
The implementation can define extra macros that begin with **SIG_**.

If an application needs to use any function that the POSIX Standard defines, it should contain the following line before any **#include** directives:

```
#define _POSIX_SOURCE 1
```

This sets the **_POSIX_SOURCE** feature-test macro. If this is done, the POSIX Standard reserves symbols for some header files. If an application includes one of the following header files, it must *not* use any of symbols reserved for that header:

<dirent.h>
Reserved prefix: **d_**.

<fcntl.h>
Reserved prefixes: **l_**, **F_**, **O_**, and **S_**. Reserved symbols: **SEEK_CUR**, **SEEK_END**, and **SEEK_SET**.

<grp.h>
Reserved prefix: **gr_**.

<limits.h>
Reserved suffix: **_MAX**.

<pwd.h>
Reserved prefix: **pw_**.

<signal.h>
Reserved prefixes: **sa_**, **SIG_**, and **SA_**.

<sys/stat.h>
Reserved prefixes: **st_** and **S_**.

<sys/times.h>
Reserved prefix: **tms_**.

If an application **#include**s any header described in the POSIX Standard, all symbols with the suffix **_t** are reserved.

Note that the symbols defined above that begin with an upper-case letter may be used by an application after the **#include** directive if the application uses an **#undef** directive to cancel any conflicting definition supplied by the header.

Example

The following program illustrates the concept of name space. It shows how the identifier **foo** can be used numerous times within the same scope yet still be distinguished. This is extremely poor programming style. Please do not write programs like this.

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

/* structure tag */
struct foo {
      /* structure member */
      struct foo *foo;
      int bar;
};

main()
{
      /* ordinary identifier */
      struct foo *foo;
      int i = 0;

      foo = (struct foo *)malloc(sizeof(*foo));
      foo->bar = ++i;
      foo->foo = NULL;

/* label */
foo:  printf("What kind of \"foo\" am I?\n");
      if (foo->foo == NULL) {
            foo->foo = (struct foo *)malloc(sizeof(*foo));
            foo->foo->foo = NULL;
            foo->foo->bar = ++i;
            goto foo;
      }

      printf("The foo loop executed %d times\n", foo->foo->bar);
      return(EXIT_SUCCESS);
}
```

See Also

C language
ANSI Standard, §3.1.2.3

Notes

Pre-ANSI implementations disagree on the name spaces of structure/**union** members. The Standard adopted the “Berkeley” rules, which state that every unique structure/**union** type has its own name space for its members. It rejected the rules of the first edition of *The C Programming Language*, which state that the members of all structures and **union**s reside in a common name space.

named pipe — Definition

A *named pipe* is a special file created with the command **/etc/mknod**. Unlike the block- and character-special files created by **mknod**, a named pipe is not a device.

A named pipe acts like a conventional pipe set up between related processes. It differs in that it has a visible name that can be seen in a file system. It also differs in that it has permissions (since it’s a file and has a name) associated with it just like any other file. This allows a named pipe to be accessed by processes that are *not* related to each other, and can even be used for processes that are running on behalf of different users.

The following illustrates how one process can write data into a named pipe and an unrelated process can read from it:

```
/etc/mknod my_pipe p                 # create the named pipe
chmod 644 my_pipe
ls -lR / > my_pipe &                 # pump data into pipe in background
mail fred < my_pipe                  # read from the pipe and process
```

This script creates a named pipe called **my_pipe** and makes sure that it is readable; it then pumps a mass of data into the pipe (in the background), and finally has a process read data from the named pipe and perform some action on them (in this case, mail the data to user **fred**). In this example, the **mail** process could be running from another login and could either be in the foreground or background.

See Also

libsocket, mkfifo(), mknod, pipe, Using COHERENT
POSIX Standard, §5.4.2

nap() — System Call (libc)

Sleep briefly
long nap(*interval***)**
long *interval***;**

nap() sleeps for *interval* milliseconds, or until its process receives a signal, whichever occurs first.

If it receives no signal, **nap()** returns the number of milliseconds it slept. If it received a signal, it returns -1 and sets **errno** to **EINTR**.

See Also

libc, sleep()

Notes

nap() is governed by the granularity of the system clock. Under COHERENT, the system clock ticks every ten milliseconds; thus, the call **nap(1);** and the call **nap(9);** have the same effect. Note that **nap()** is guaranteed to sleep for at least *interval* milliseconds; thus, the call **nap(11);** sleeps for two clock ticks, or 20 milliseconds.

ncheck — Command

Print file names corresponding to i-node
ncheck [-i *number* **...] [-as]** *filesystem ...*

An *i-number* identifies an i-node. **ncheck** generates a list of file names by i-number for each *filesystem*, which should be the name of a device special file that contains a proper COHERENT file system. Using the raw device generally decreases the time **ncheck** requires to do its work.

The output is in the unsorted traversal order of the file system hierarchy. **ncheck** distinguishes directories from files by suffixing '/.' to directory names.

Under the **-i** option, **ncheck** prints the file name corresponding to each i-number *number* in the given list. Under the **-a** option, **ncheck** prints only the names of special files and set user-ID mode files; this option allows the system administrator to ascertain quickly whether these files represent possible security breaches.

See Also

commands, i-node

Diagnostics

ncheck appends '??' to the generated file name if it cannot find the proper parent structure while retrieving the file-name information. It represents any loops detected in the file name by the characters '...'. Extremely addled file systems may generate other reasonably self-explanatory diagnostics.

ndbm.h — Header File

Header file for NDBM routines
#include <ndbm.h>

Header file **<ndbm.h>** declares the functions used to manipulate NDBM data bases:

dbm_clearerr()	Clear an error condition on an NDBM data base
dbm_close()	Close an NDBM data base
dbm_delete()	Delete records from an NDBM data base
dbm_dirfno()	Return the file descriptor for an NDBM .dir file
dbm_error()	Check a NDBM data base for an error
dbm_fetch()	Fetch a record from an NDBM data base
dbm_firstkey()	Retrieve the first key from an NDBM data base
dbm_nextkey()	Retrieve the next key from an NDBM data base
dbm_open()	Open an NDBM data base
dbm_pagfno()	Return the file descriptor for an NDBM .pag file
dbm_rdonly()	Set an NDBM data base into read-only mode

dbm_store() . . . . Store a record into an NDBM data base

Routines **dbm_error()** and **dbm_clearerr()** are macros that, in fact, do nothing.

This header file also defines two structures that the NDBM routines use. The first, **datum**, defines the structure of a data element, either a key or its associated data set:

```
typedef struct {
      char *dptr;
      int dsize;
} datum;
```

This structure lets you have a key and a data element of unlimited length.

The other structure, **DBM**, holds the information that the NDBM routines use to access a NDBM data base:

```
typedef struct {int dummy[10];} DBM;
```

See Also

Notes

For a statement of copyright and permissions on this header file, see the Lexicon entry for **libgdbm**.

netdb.h — Header File

Define structures used to describe networks
#include <netdb.h>

Header file **<netdb.h>** defines structures into various **sockets** functions write information about the local network. It also defines manifest constants and macros used by various **sockets** routines.

See Also

endnetent(), endprotoent(), endservent(), getnetbyaddr(), getnetbyname(), getnetent(), getprotobyname(), getprotobynumb(), getprotoent(), getservbyname(), getservbyport(), getservent(), header files, libsocket, setnetent(), setprotoent(), setservent()

networks — System Administration

Name remote networks
/etc/networks

The file **/etc/networks** names remote networks with which you can communicate, and gives information with which your system can pass datagrams to those networks.

If you wish to communicate on the Internet, you must create this file by obtaining the official network data base maintained by the Network Information Control Center (nic.ddn.mil). To this, add information about other networks not listed by NIC, with which you may wish to communicate.

If you are not going to use the Internet, you can create your own version of **/etc/networks**. Each line within **networks** describes one remote network, and consists of the following fields:

- The network's name. A network name can contain any printable character other than white space, a newline character, or the comment character '#'.
- The network's Internet-protocol (IP) address, in standard dot notation.
- Aliases, if any, for the network's name.

For example:

```
mysubnet    127.0.1          an_alias    # a comment
```

If you create your own version of **/etc/networks**, be sure to set its permissions correctly. It should be owned by the superuser **root**, and be executable.

See Also

Administering COHERENT, hosts, hosts.equiv, inetd.conf, protocols, services

newaliases — Command

Build the smail aliases data base from an ASCII source file
/usr/lib/mail/newaliases

Command **newaliases** reads the ASCII source file for an aliases data base, and builds the aliases data base according to the configuration information in **/usr/lib/mail/config**. Run this program whenever changes have been made to the ASCII source file. If this program is not used, **smail** may not notice the changes that have been made.

The aliases data base can be in a DBM data base, a sorted text file, or a plain text file. (For information about what a DBM data base is, see the Lexicon entry for **libgdbm**.) In the latter case, which is the default under COHERENT, the ASCII source file doubles as the data-base file.

To process an file, first use the command **mkline** to remove comments and regularize it. If you wish to build a sorted data base, then use the command **mksort** with its command-line option **-f** to create the sorted data base. If you wish, however, to build a DBM data base, use command **mkdbm**, again with option **-f**, to create the data base. In either case, be careful that **smail** never uses a truncated or partially built data base.

For plain text data bases, **newaliases** displays a summary of its contents, but no changes are actually made.

Files

/usr/lib/mail/aliases
The text file that defines aliases.
/usr/lib/mail/aliases.dir
/usr/lib/mail/aliases.pag
The DBM data base that is built from the text file **aliases**.
/usr/lib/mail/config
The file that gives the default configuration for **smail**.

See Also

commands, libgdbm, mail [overview], mkdbm, mkline, mksort, smail

Notes

The name **newaliases** is retained for compatibility with BSD **sendmail**. Under **smail** release 3.1, this command usually is named **mkaliases**.

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**.

newgrp — Command

Change to a new group
newgrp *group*

newgrp changes the user's group identification to the specified *group*, if access is permitted. The file **/etc/group** determines group access. Group access may be unrestricted, or open to all users with specific exceptions, or restricted to certain users via a password.

The shell executes **newgrp** directly.

Files

/etc/group — Give group access

See Also

commands, group, ksh, sh

Diagnostics

If **newgrp** succeeds, no diagnostic is printed.

Notes

Interruption of **newgrp** can result in the user being logged off.

Under the Korn shell, **newgrp** is an alias for **exec newgrp**.

newusr — Command

Add new user to COHERENT system
/etc/newusr *login "User Name" parentdir* **[** *shell* **]**

newusr adds a new user to the system. It automatically adds an entry to the file **/etc/passwd**, creates a home directory for the user, installs the user in the mail system, and otherwise performs the myriad tasks required to add a new user to your COHERENT system.

login is the login idenifier of the new user. This is a single word in lower case, by which that user is identified. Note that each user must have a unique login identifier. Identifiers are usually the user's first name, initials, or a nickname. *parentdir* is the directory or (more usually) the file system in which **newusr** will create the new user's home directory. *User Name* is the name of the human for whom *login* is being created. *shell* names the shell to be used; the default is the Bourne shell **/bin/sh**.

For example, the command

```
/etc/newusr batman "Bruce Wayne" /v /usr/bin/ksh
```

creates new user Bruce Wayne, with login **batman**, home directory **/v/batman**, and default shell **/usr/bin/ksh**.

Files

/etc/group — User groups
/etc/passwd — User passwords
/*parentdir*/*user* — User home directory
/usr/spool/mail/*user* — User mailbox

See Also

commands, passwd, welcome

Diagnostics

newusr complains if an entry for *user* already exists in the password file.

Notes

Only the superuser can add new users to the system with **newusr**.

nextkey() — DBM Function (libgdbm)

Retrieve the next record from a DBM data base
#include <dbm.h>
datum nextkey ()

Function **nextkey()** retrieves the next record from the currently open DBM data base. The data base must first have been opened by a call to **dbminit()**, and the first record within the data base must have been retrieved by a call to **firstkey()**.

nextkey() returns a pointer to the retrieved record. If no record is available (i.e., every record has already been retrieved), or if an error occurred, field **dptr** within the returned record is initialized to NULL.

You can use this function with function **firstkey()** to walk through the entire contents of a DBM data base. For example:

```
for(key=firstkey(); key.dptr!=NULL; key=nextkey(key))
```

Please note that the hashing algorithm used the DBM functions dictates which record is "next" within the data base. A loop that uses this function plus the function **firstkey()** will retrieve every record from the data base; however, the records probably will not be in the order you expect.

See Also

Notes

For a statement of copyright and permissions on this routine, see the Lexicon entry for **libgdbm**.

nm — Command

Print a program's symbol table
nm [-**adgnopru**] *file* ...

The command **nm** prints the symbol table of each *file*. It can read binary files produced by the compiler, assembler, or linker.

When a C source file is compiled with the **-c** switch to the **cc** command, or when a file of assembly language is assembled, the result is an object module, which is signified by the suffix **.o**.

The linker **ld** links multiple object modules to form an executable program. Frequently used object modules often are grouped by the archiver **ar** into a *library*, which is signified by the suffix **.a**. **nm** can read all three kinds of files: .o, .a, and fully linked executables.

Options

nm recognizes the following options:

-a (COHERENT 286 only)
Print all symbols. Normally, **nm** prints names that are in C-style format and ignores symbols with names inaccessible from C programs.

-d Print only defined symbol.

-g Print only global symbols.

-n Sort numerically rather than alphabetically. **nm** uses unsigned compares when sorting symbols with this option.

-o Append the file name to the beginning of each output line.

-p Print symbols in the order in which they appear within the symbol table.

-r Sort in reverse-alphabetical order.

-u Print only undefined symbols.

Output Format

The output of **nm** is a series of lines of the form:

segment address symbol

segment gives the segment in which the symbol appears, or **UNDEF** for undefined symbols. *address* is either the address in hexadecimal, or the length of a common variable. *symbol* names the symbol.

For example, if **foo.o** is a relocatable object module, the output of the command **nm -o foo.o** would appears as follows:

```
#nm foo.o
UNDEF    00000000 _canl
UNDEF    00000000 _stderr
.text    0000077C acomp
.text    00000034 acomp_old
UNDEF    00000000 alloc
.text    00000F28 archive
.comm    00000004 asw
.text    000003CC csymbol
.comm    00000004 dsw
```

See Also

cc, commands, ld, size, strip

nohup — Command

Run a command immune to hangups and quits
nohup *command* [*arguments*]

The command **nohup** tells the COHERENT shell to execute *command* while ignoring all hangup and quit signals.

If you do not redirect the output of *command*, **nohup** redirects both the standard output and the standard error into the file **nohup.out**. If **nohup.out** cannot be created in the current directory, **nohup** redirects all output into the file **$HOME/nohup.out**.

nohup is often used to execute scripts or pipelines that would normally abort if you logged out during the middle of execution.

Examples

If **file** is a shell script, then the command

```
    nohup sh file
```

executes the contents of **file** in the foreground while ignoring all quit or hangup signals. The command

```
    nohup sh file &
```

executes **file** in the background; you can log out safely and all the contents of **file** will still be executed.

See Also

commands, kill, ksh, sh, signal()

nologin — System Administration

Lock out logins
/etc/nologin

login looks for file **/etc/nologin** before it permits a user to login in. If this file exists, **login** forbids the user to log in, and instead displays on the terminal the contents of this file — which, presumably, explain why logging in is now forbidden.

You should create this file when you wish to “lock out” users during a critical time, such as when backups are being run or when the system is about to be shut down. When the critical time has passed, be sure to remove it.

login cannot lock out the superuser **root**, even if **nologin** exists. Nor will it lock out the users named in the file **/etc/trustme**, should it exist.

See Also

Administering COHERENT, login, trustme

Notes

The script **/etc/rc** removes **/etc/nologin** by default, on the assumption that after you reboot, you once again want users to be able to log in. If this is not a sound assumption, edit **/etc/rc** to change this behavior.

notmem() — General Function (libc)

Check whether memory is allocated
int notmem(*ptr***);**
char ****ptr*;**

notmem() checks if a memory block has been allocated by **calloc()**, **malloc()**, or **realloc()**. *ptr* points to the block to be checked.

notmem() searches the arena for *ptr*. It returns one if *ptr* is not a memory block obtained from **malloc()**, **calloc()**, or **realloc()**, and zero if it is.

See Also

arena, calloc(), free(), libc, malloc(), memok(), realloc(), setbuf()

Notes

The only valid use for **notmem()** is in debugging code, such as in the bodies of calls to the macro **assert()**. We furthermore recommend that portable code should conditionalize use of **notmem()** so that the code may continue to compile on systems that lack such a facility.

nptx — Command

Generate permutations of users' full names
/usr/bin/nptx

The command **nptx** reads an address/name pair (that is, an address and a user's full name), and prints on the standard output as many permutations of the user's name as it can devise, each linked to the given address. A set of such permutations helps to relieve a user of the need to know the exact form of another user's name when she wishes to send mail to that user. When a set of users' names is filtered through **nptx**, the mail program **smail** can use the output as a "full-name data base".

The format of an input line is:

```
	name<tab>address
```

name gives the user's first name, last name, optional middle initial, and optional nickname in parentheses; all are separated by space characters. *address* can contain any e-mail address. *name* and *address* are separated by one **<tab>** character.

nptx prints all permutations of the first names and initials, with the last name appearing in each permutation. Permutations are not necesarily unique.

Example

Given the name/address pair

```
	LaMonte Cranston(Shadow)<tab>shadow@goodguy.com
```

nptx produces the following set of permutations:

```
	Cranston               shadow@goodguy.com
	L.Cranston             shadow@goodguy.com
	LaMonte.Cranston       shadow@goodguy.com
	S.Cranston             shadow@goodguy.com
	Shadow.Cranston        shadow@goodguy.com
```

See Also

commands, mail, mkfnames, paths, smail

Notes

nptx normally is invoked via the script **mkfnames**, which reads a file of names (or the file **/etc/passwd** and generates a data base of names and addresses that can be used by the mail system.

nptx assumes European-style names, i.e., that the family name comes last (unlike Asian or Hungarian names, in which the family name comes first).

nrand48() — Random-Number Function (libc)

Return a 48-bit pseudo-random number as a non-negative long integer
long nrand48(*xsubi***)**
unsigned short *xsubi***[3];**

Function **nrand48()** generates a 48-bit random number, then returns its high 31 bits in the form of a non-negative **long**. The value returned is (or should be) uniformly distributed throughout the range of zero through 2^31. *xsubi* is an array of three unsigned short integers from which the pseudo-random number is built.

See Also

libc, srand48()

nroff — Command

Text-formatting language
nroff [*option ...***] [***file ...***]**

nroff is the COHERENT text-formatter and text-formatting language. By embedding commands within files of text, you can instruct **nroff** to format text, create paragraphs, subheadings, headers, footers, and in general perform all tasks required to format text for the printed page or for screen display.

nroff is designed to be used with character-display terminals or monospace printers. The related program **troff** performs typeset-quality formatting, suitable for printing on the Hewlett-Packard LaserJet printer or any printer for which the PostScript language has been implemented. **troff**'s formatting language is a superset of that used by **nroff**. Text that you have encoded for formatting by **nroff** will work with **troff**, but the reverse is not always true. See the Lexicon entry on **troff** for information that applies to **troff** alone.

nroff Input

nroff processes each *file*, or the standard input if none is specified, and prints the formatted result on the standard output. The input must contain formatting instructions as well as the text to be processed.

Basic **nroff** commands provide for such things as setting line length, page length, and page offset, generating vertical and horizontal motions, indentation, filling and adjusting output lines, and centering. The great flexibility of **nroff** lies in its acceptance of user-defined macros to control almost all formatting. For example, the formation of paragraphs, header and footer areas, and footnotes must all be implemented by the user via macros.

The following summarizes the commands and options that can be used with **nroff**. Four types of commands and options are described: (1) command line options; (2) **nroff**'s basic commands (also called *primitives*); (3) escape sequences that can be used with **nroff**; and (4) **nroff**'s dedicated number registers, and what information each one keeps.

Command-line Options

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

-d Debug: print each request before execution. This options is extremely useful when you are writing new macros.

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

-k Keep: do not erase the temporary file.

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

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

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

-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. Use this option when you wish to use **nroff** interactively. It, too, is useful when debugging macros.

nroff appends the contents of the environmental variable **NROFF** to the beginning of the list of command-line arguments. This let you set commonly used options once in the environment, rather than retype them for each invocation of **nroff**.

Primitives

The following gives the basic commands, or *primitives*, that are built into **nroff**. These primitives can be assembled into macros, or can be written directly into the text of your document. Commands may begin either with a period '.' or with an apostrophe; the former causes a break (see **.br**, below), the latter does not.

.ab *msg*
Abort: print *msg* on the standard error and abort processing.

.ad [bclr]
Enter adjust mode: that is, insert white space between words to create right-justified output. **b** adjusts for both margins; this is the default. **c** adjusts and centers on the line. **l** adjusts, flush with the left margin. **r** adjusts, flush with the right margin.

.af *R X* Assign format *X* to number register *R*. The assigned format may be one of the following:

1	Arabic numerals (default)
i	Lower-case Roman numerals
I	Upper-case Roman numerals
a	Lower-case alphabetic characters
A	Upper-case alphabetic characters

.am *XX* Append the following to macro *XX*. Used like **.de**, below.

.as *XX* Append the following to string *XX*. Used like **.ds**, below.

.bp Begin a new page.

.br Break; print any fraction of a line of text that is in the input buffer before reading new text.

.c2 *c* Set the no-break control character to *c*. With no argument, reset it to the default character, which is the apostrophe.

.cc *c* Set the normal control character to *c*. With no argument, reset it to the default character, which is the period.

.ce *N* Center *N* lines of text (default, one).

.ch *XX N*
Change the location of the trap for macro *XX* to vertical position *N* on the page. Used like command **.wh**, below.

.co *endmark*
Copy input directly to the output until *endmark* is seen. If no *endmark* is given, copy until another **.co** is seen.

.cu *N* Underline the next *N* lines. When used without an argument, one line is underlined. The instruction

```
.cu 0
```

turns off underlining. Note that unlike the UNIX version of **nroff**, **.cu** does not perform continuous underlining — it underlines words, but not spaces.

.da *X* Divert and append the following text into macro *X*. A diversion is ended by a **.da** command that has no argument.

.de *X* Define macro *X*. The macro definition is ended by a line that contains only two periods "..".

.di *X* Divert the following text into macro *X*. Diversion is ended by a **.di** command that has no argument.

.ds *X value*
Define string *X* to have the given *value*.

.ec *c* Set the escape characer to *c*. With no argument, reset it to the default backslash character '\'.

.el *action*
Execute *action* when the test in an **.ie** command fails. This command must be used with an **.ie** command.

.em *XX* Execute macro *XX* when processing is completed.

.eo Escape off: turn off special handling of all escape sequences.

.ev *N* Change the environment. When followed by 0, 1, 2, the command *pushes* that environment; when used without an argument, the command *pops* the present environment and returns to the previous environment.

.ex Exit from **nroff** without further ado.

.fi Enter fill mode.

.fl Flush; same as **.br**.

.ft *X* Change the current font to *X*. **nroff** recognizes **R**, **B**, and **I**, for Roman, bold, and italic, respectively.

.ie *condition action*
This command tests to see if *condition* is true; if true, it then executes *action*; otherwise, it performs the action introduced by an **.el** primitive. This command must be used with the **.el** command.

.if *condition action*
This command tests to see if *condition* is true; if so, then *action* is executed; otherwise, *action* is ignored. The command **.if o** applies if the page number is odd, and the command **.if e** applies if the page number is even. The command **.if n** applies if the text is processed by **nroff**, and the command **.if t** applies if the text is processed by **troff**. The command **.if l** applies in landscape mode. The command **.if p** applies to **troff** PostScript mode. Note that the last two conditions are unique to the COHERENT implementation of **nroff**, and may not be portable to other implementations.

.ig *X* Ignore all input until macro *.X* is called; if no argument is given, ignore input until two periods "..".

.in *NX* Change the normal indentation to *N* units of *X* scale. *X* can be **u** or **i**, for *machine units* or *inches*, respectively. If *N* is used without *X*, **nroff** assumes the indentation to be given in number of character-widths (in picas, or tenths of an inch). Default indentation is zero.

.it *N XX*
Set an input trap to execute macro *XX* after *N* input lines (not counting request lines).

.lc *c* Set the leader dot character to *c*. When **nroff** sees the escape sequence **\a**, it fills space to the next tab stop with the leader dot character. **lc** with no argument tells **nroff** to use spaces to fill leaders.

.ll *NX* Set the line length. Used like the **.in** command, above.

.ls *X* Leave spaces; insert *X* vertical spaces after each line of text. Default is zero.

.lt *NX* Length of title. Used like the **.in** command, above.

.na Enter no-adjust mode. Line lengths are not changed.

.ne *NX* Confirm that at least *N* portions of *X* units of measure of vertical space are needed before the next trap. If this amount of space is not available, then move the text to the top of the next page. *X* can be **i** or **v**, for inches or vertical spaces, respectively. This command is used in display macros and in paragraph macros to help prevent widows and orphans.

.nf Enter no-fill mode; no right justification is performed, although line lengths are changed to approximate uniform line length.

.nh Turn off hyphenation. **nroff** hyphenates according to built-in algorithms that are correct most of the time, but not always.

.nr *X N1 N2*
Set number register *X* to value *N1*; set its default increment/decrement to *N2*. For example, **.nr X 2 3** sets number register **X** to 2, and sets its default increment to 3.

The basic unit of measurement for **nroff** 1/120th of an inch; this is also called the *machine unit*. It is indicated by the suffix **u** to a measurement. Unless otherwise stated, all number registers that information about a page holds that information in **nroff** machine units.

Other units of measure convert into **nroff** units as follows:

inch:	1**i** = 120**u**
vertical line space:	1**v** = 20**u**
centimeter:	1**c** = 47**u**
em:	1**m** = 12**u**
en:	1**n** = 12**u**
pica:	1**P** = 20**u**
point:	1**p** = 1**u**

.ns No-space mode.

.nx *file* Terminate processing of the current input file and begin processing *file* instead.

.pl *NX* Set the page length to *N*. The unit of measure *X* can be **V** or **i**, for vertical spaces (sixths of an inch) or inches, respectively. The default unit of measure is vertical spaces.

.pn *N* Set the page number to *N*.

.po *NX* Set the default page offset to *N*. The unit of measure *X* can be set to **i**, for inches. The default unit of measure is number of characters.

.rb *file* Read binary: read the given *file* and copy it directly to the output without processing.

.rd *prompt*
Read an insertion from the standard input after issuing the given *prompt*.

.rf *XX YY*
Rename font *XX* as *YY*. For example, to have calls to font K remapped to Roman font, use the call:

```
.rf K R
```

.rm *XX* Remove macro or string *XX*.

.rn *XX YY*
Change the name of a macro or string from *XX* to *YY*.

.rr *X* Remove register *X*.

.rs Restore normal space mode.

.so *file* Open *file*, read its contents, and process them. When the end of *file* is reached, resume processing the contents of the present file.

.sp [|]*NX*
Space down *N*. The unit of measure *X* can be **i**, for inches, with the default unit of measure being vertical spaces, or sixths of an inch. The optional vertical bar '|' indicates that *N* is an absolute value; for example, **.sp |1.5i** means to move to 1.5 inches below the top of the page, whereas **.sp 1.5i** means to move to 1.5 inches below the present position.

.sy *command*
Execute *command* under the shell. Please note that this primitive is non-standard. Macros that use it cannot be formatted under standard AT&T **nroff**.

.ta *NX ...*
Set the tab to *N*. The unit of measure *X* can be set to **i**, for inches; the default unit of measure is number of characters, or tenths of an inch. A tab setting, of course, is for an absolute, not a relative, value. If more than one tab setting is defined, the first defines the first tabulation character on a text line, the second defines the second tabulation character, etc. Any undefined tabulations are thrown away.

.tc *X N* Fill any unused space within a tabulation field with the character *X*. If the optional *N* is present, it specifies a width for the character; for example, **.tc . .1i** fills tabs with dots spaced one-tenth of an inch apart.

.ti *NX* Temporary indent; indent only the next line. Used like the **.in** command, above.

.tl *'left'center'right'*
Set a three-part title, with *left* being set flush left, *center* being centered on the line, and *right* being set flush right. Note the use of the apostrophes to separate the fields; the apostrophes for an undefined field must still be present, or a syntax error will be generated.

.tm *message*
Print *message* on the standard error device. This is often used with **.if** or **.ie** commands to indicate an error condition.

.tr *xy* Translate character *x* to *y* on output.

.ul *N* This behaves the same as **.cu**.

.vs *Np* Reset the normal vertical spacing to *N* points **p**. One point equals 1/72 of an inch. The default setting one pica, which equals is 12 points or 1/6 of an inch.

.wh *NX action*
Set a trap to perform *action* when point *N* is reached on every formatted page. If *N* is negative, it is measured up from the bottom of the page. The unit of measure *X* may be **i** or **v**, for inches or number of vertical lines, respectively; the default unit of measure is **v**.

Escape Sequences

The following lists **nroff**'s escape sequences, or commands that suspend or work around the normal operation of **nroff**. Each escape sequences is introduced by the *escape character*, normally the backslash character '\':

\(*xx* Print special character *xx*, as defined by a **.dc** request. **nroff** reads default special character definitions from file **/usr/lib/roff/nroff/specials.r**. For example, the escape sequence **\(<=** prints the less-than-or-equal-to symbol ≤.

\. Print a literal period.

\' Print a literal apostrophe. This should be used in text that will be manipulated by the **\w** escape sequence or the **.tl** primitive.

\\ Delay interpretation of a backslash character. This normally is used to defer the interpretation of a macro or string from the time it is processed to the time that it is called.

\- Print a minus sign.

\& Ignore what is normally a command string.

\$*N* Call macro argument *N*.

\" Introduce a comment within your text. All text to the right of this escape sequence will be ignored by **nroff**. This sequence must read **.\"** when used at the beginning of a line.

*****S* Call string *S*.

(ST* Call string *ST*.

\a Fill the space to the next tab stop with leader dots (normally '.').

\d Move *down* by one-half em (**troff**) or one-half line (**nroff**). Normally used to do crude subscripting, or to undo the effect of the **\u** escape sequence.

\e Print the escape character in the output text — normally, a backslash.

\f*X* Set font to *X*; this can be either **R**, **I**, **B**, or **P**, for Roman, *italic*, **bold**, or previous font, respectively.

\h'[|]*NX*'
Move horizontally by *N* units of *X*. If *N* is positive, move to the right; if negative, move to the left. The unit of measure *X* may be **i**, for inches; the default unit of measure is ems. (One em equals one pica, which is one-sixth of an inch). When the optional vertical bar '|' is used, move to an absolute position on the line. For example **\h'|1.5i'** moves to 1.5 inches to the right of the left margin, whereas **\h'1.5i'** moves 1.5 inches to the right of the current position.

\k*x* Record the current vertical position into register *x*.

\l'*NX*' Draw a horizontal line *N* units of *X* long. The unit of measure *X* may be **i**, for inches; the default unit of measure is character-widths.

\L'*NX*' Draw a vertical line; used like **\l**, above.

\n*X* Read the value of number register *X*.

\n(*XY* Read the value of number register *XY*.

\o'*chars*'
Overstrike the given *chars*, centered on the widest.

\s*N* Change the current size of the type to *N* points.

\s+*N* Increment the current point size by *N* points.

\s-*N* Decrement the current point size by *N* points.

\t Print a tab.

\u Move *up* by one-half em (**troff**) or one-half line (**nroff**). Normally used to do crude superscripting, or to reverse the effect of the **\d** escape sequence.

\v'*NX*' Vertical motion; move *N* units of *X* vertically. If *N* is positive, move down; if negative, move up. The unit of measure *X* may be **i** or **v**, for inches or vertical spaces (sixths of an inch), respectively. The default unit of measure is **v**.

\w'*argument***'**
Measure the width of *argument*. For example

```
    \w'stuff and nonsense'
```

measures the width of the phrase **stuff and nonsense**; or

```
    \w'\$1'
```

measures the width of the first argument passed to a macro, whatever that argument might happen to be. Therefore, the command **.in \w'\$1'** will indent a line by the width of argument 1.

\X*dd* Output the character with hexadecimal value *dd*, where *dd* are two hexadecimal digits. Users can use this option to encode characters that are not part of the English-language character set. The hexadecimal values to which characters map depend upon the character set that you (or your printer) use. Please note **nroff** reserves the following values for its internal use:

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

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

\z*c* Print character *c* with zero width.

\<newline>
Ignore this **<newline>** character.

\{ Begin conditional commands; used after an **.if**, an **.ie**, or an **.el** command.

\{ Begin conditional commands, and ignore the following carriage return.

\} End conditional commands.

Dedicated Number Registers

The following lists the number registers that are predefined in **nroff**. You can read or reset these registers to suit the need of any special formats that you wish to devise.

$$ Process identifier of the current **nroff** process. This usually is used with the primitive **.sy** to name temporary files.

.$ Number of arguments passed to a macro.

% Present page number.

.c Number of lines read from the current input file. This can be used to help set an input-line trap.

.d Current vertical position in the current diversion. If no diversion is opened, this register's contents equal those of the **nl** register, described below.

dl Maximum width of last completed diversion.

dn Height of last completed diversion.

dw Day of the week (one through seven; one indicates Sunday).

dy Day of the month, as set by COHERENT.

.F Name of input file being read. This is very useful for printing error messages. This register applies only the COHERENT implementation of **nroff**. Code that uses it is not portable to other implementations.

.h Vertical position of the current line's base-line. This number register gives you the best idea of your current vertical position on the page.

hp Horizontal position on current input line.

.i Present amount of indentation.

.j	Current type and mode of text adjustment.
.l	Present line length.
ln	Current line number in the output.
mo	Month, as set by COHERENT.
.n	Width of the text portion of the previously printed line. Useful for underlining, shading, or otherwise modifying the previous line of text. For example `\l'\n(.nu'` draws a line under the previously printed line of text.
nl	Vertical position of the base-line of the last printed line of text.
.o	Present page offset.
.p	Page length.
.s	Size of the type currently being printed, in points.
sb	Depth to which a string hangs below its base line. This is generated by the width function.
st	Height to which a string extends above its base line. This is generated by the width function.
.t	Distance to the next trap. Check this register to see if the object you wish to print on a page will fit.
.v	Size of a line, in points. This is set by the **vs** primitive.
yr	Last two digits of the year, as set by COHERENT.
.z	Name of the current diversion.

Printer Configuration

nroff reads several files in directory **/usr/lib/roff/nroff** to find printer-specific information. It reads special character definitions from file **specials.r.** If file **fonts.r** exists, **nroff** reads font information from it; **nroff** understands only Roman, bold and italic fonts, but **.rf** requests may define alternative font names. If file **.pre** exists, **nroff** copies it at the beginning of the output. If file **.post** exists, **nroff** copies it at the end of the output. In landscape mode, **nroff** looks for files **.pre_land** and **.post_land** instead. You can change these files as desired to include printer-specific commands in **nroff** output.

Miscellaneous

The **-ms** macro package is kept in file **/usr/lib/tmac.s.** The macros in this package are more than sufficient for most ordinary text processing. Beginners should work through this macro package rather than trying to deal at once with the basic program.

The tutorial to **nroff**, which is included with this manual, provides a detailed introduction to **nroff**. Error messages for **nroff** appear in the appendix to this manual.

Files

/tmp/rof* — Temporary files
/usr/lib/tmac.* — Standard macro packages
/usr/lib/roff/nroff/ — Support files directory
/usr/lib/roff/nroff/.pre — Output prefix
/usr/lib/roff/nroff/.pre_land — Output prefix, landscape mode
/usr/lib/roff/nroff/.post — Output suffix
/usr/lib/roff/nroff/.post_land — Output suffix, landscape mode
/usr/lib/roff/nroff/fonts.r — Alternative font name definitions
/usr/lib/roff/nroff/specials.r — Special character definitions

See Also

col, commands, deroff, man, ms, printer, troff
nroff, The Text-Formatting Language, tutorial

Diagnostics

nroff returns the following error messages. Most are self-explanatory.

-f option requires file argument *(fatal)*
.bd not implemented yet
.co: unexpected EOF before *string (error)*
.dt not implemented yet
.el without .ie *(error)*
.fc not implemented yet
.hc not implemented yet
.hw not implemented yet
.hy not implemented yet

.ie nested more than *N* levels *(error)*
The **.ie/.el** combination can be nested only 15 levels deep.

.ie without matching .el *(error)*
Every **.ie** must be followed by an **.el**.

.lf: *string*, file "*string*" *(error)*
troff could not load a font-width table from file *string*.

.lf: "*string*" is not a PCL font width table *(error)*
troff expects a PCL font-width table, but file *string* is not in the PCL font-width format.

.lf: "*string*" is not a PostScript font width table *(error)*
troff expects a PostScript font-width table, but file *string* is not in the PostScript font-width format.

.lf: cannot load more than *N* fonts *(error)*
troff has a static limit on the number of font-width tables that can be loaded at one time.

.lf: cannot open file "*string*" *(error)*
.lf: requires fontname and filename *(error)*
.nm not implemented yet
.nn not implemented yet
.pi not implemented yet
.rb: cannot open file *string (error)*
.rb: no file specified *(error)*
.rf: requires name and new name *(error)*

\} without matching \{ *(error)*
Every **\}** must be preceded by a **\{**.

arguments too long *(error)*
attempted zero divide *(error)*
attempted zero modulus *(error)*
bad adjustment type *(error)*
bad argument reference *(error)*
bad directive *N (fatal)*
bad font *N (fatal)*
bad font *N* at dev_font, nfonts=*N (fatal)*
bad font *N*, nfonts=*N (fatal)*
bad pattern *(fatal)*
bad tab stop *(error)*
bad tab stop *(error)*

botch: fontname(*N*) *(fatal)*
nroff cannot handle font *N* and must abort processing.

botch: swdmul=*N* psz=*N* swddiv=*N (fatal)*
An undefined error has occurred within **nroff**. The printed numbers give the value of **nroff**'s internal registers. If such an error occurs regularly when you process a given piece of text, please send the text in question and a copy of the error message to Mark Williams technical support.

bracket building not implemented yet
cannot create temp file *(fatal)*
cannot dehyphenate *(fatal)*

cannot end diversion *(error)*
    You attempted to close a diversion without first opening one.
cannot find current file *(error)*

cannot find font *XX (error)*
    Font *XX* has not been opened; therefore **[nt]roff** cannot use it. To open a font, use the load-font primitive **.lf**.

cannot find font *N (error)*

cannot find register *string (error)*
    You attempted to read a number register without first loading a value into it.

cannot open *string (error)*
cannot open file "*string*" *(error)*

cannot pop environment *(error)*
    You popped an environment without first pushing one.

cannot read environment *(fatal)*
cannot remove *string (error)*
cannot reopen temp file *(fatal)*
cannot write environment *(fatal)*
delimiter argument too large *(error)*
diversion buffer odd alignment *(fatal)*
environment does not exist *(error)*
environments stacked too deeply *(error)*
field with too large *(error)*
file "*string*" not found *(error)*
flushd -- current diversion null *(fatal)*
font position out of range *(error)*

fonts.r not found *(fatal)*
    nroff and **troff** read the list of fonts to use from a file named **fonts.r**. If you do not have such a file in your current directory, **nroff** and **troff** read the one out of their home directories: **/usr/lib/roff/nroff**, **/usr/lib/roff/troff_pcl**, or **/usr/lib/roff/troff_ps**, depending which variety of output you have requested. This error message means that your current directory does not hold a file named **fonts.r**, and that **[nt]roff** cannot open the **fonts.r** file in its appropriate home directory.

illegal hex digit *(error)*
    The escape sequence **\X***NN* prints a character by its literal hexadecimal value. This should be used when processing characters that are not normally printable on the terminal screen. Digit *N* can be the numerals '0' through '9', the letters 'a' through 'f', or the letters 'A' through 'F'. All other characters will trigger this error.

illegal option: *string (fatal)*

incomplete macro in trap *(fatal)*
    A trap has jumped to a macro, but that macro does not terminate, for whatever reason. Usually this indicates that you have opened a diversion but failed to close it.

line buffer overflow *(fatal)*
no room for new font name *XX (error)*
out of space - memory *string (fatal)*
request '*string*' not found *(error)*
section *N* of title too large *(error)*
special character *XX* not found *(error)*

syntax error *(error)*
    This message any number of errors with your **nroff** source. Check the line number given in the message.

temporary file write error *(fatal)*
    nroff cannot write a temporary file, for whatever reason. This usually indicates that you lack permission to write into the directory into which **nroff** is attempt to write its temporary files.

too many tab stops *(error)*
: **nroff** allows a maximum of nine tab stops in one line. It ignores all tab stops that exceed that limit.

unexpected end of file *(fatal)*
: This error indicates that **nroff** is in the middle of processing a macro when the file ends. This error usually occurs when you open a diversion and fail to close it.

unknown macro/register type *N (fatal)*
vertical line drawing not implemented yet *(error)*
word buffer overflow *(fatal)*

NUL — Definition

NUL is the ASCII null character '\0' — i.e., the character with the value zero. Do not confuse it with the null pointer NULL or with the empty string "". A C-language string is always terminated with a NUL. The empty string "" is an array of **char**s with only one element, namely a NUL.

See Also

ASCII, NULL, Programming COHERENT

NULL — Manifest Constant

The manifest constant **NULL** is defined in the header file **stddef.h**. It is the null pointer **(char *)0**, which is a pointer initialized to zero. Numerous routines return this value to indicate failure; it is useful as a return value because it points nowhere, and so removes the possibility of accidentally destroying a section of memory after failure.

See Also

manifest constant, NUL, pointer, stdio.h
ANSI Standard, §7.1.6

null — Device Driver

The 'bit bucket'

All data written to the special file **/dev/null** are thrown away (sent to the "bit bucket"). This is useful, for example, when you wish to test a program's side effects while ignoring its output.

A read from file **/dev/null** returns end of file (zero bytes of data). The shell **sh** uses **/dev/null** as input to background processes.

Files

/dev/null

See Also

device drivers, idle, ksh, sh

nybble — Definition

A **nybble** is four bits, or half of an eight-bit byte. The term is generally used to refer to the low four bits or the high four bits of a byte. Thus, a byte may be said to have a "low nybble" and a "high nybble". One nybble encodes one hexadecimal digit.

See Also

bit, byte, Programming COHERENT