NAME

masterfile – Internet domain name server master data file

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

A masterfile is a text file that contains authoritative data for a zone. Zones are defined in the named.boot file. A master file consists of resource records (which make up the actual data for the zone), with optional additional directives. The available directives are as follows:

$INCLUDE *masterfile* Includes the contents of *masterfile* in the interpretation of the current master file.

$ORIGIN *domain* Establishes *domain* as the default domain.

Each resource record in the master file has the following format:

[*name*] [*ttl*] *address_class* *record_type* *data*

name (Optional) Domain name being defined within the current zone. When used as names, the following symbols have special meanings:

- . Current domain
- @ Current default domain ($ORIGIN)
- .. Root domain

ttl (Optional) Time to live; a number that represents the amount of time this resource record may be considered valid by another name server that queries this name server.

address_class Class addressing used; typically, either IN (Internet class) or ANY (all classes).

record_type Type of record; some valid record types include the following:

A	Address of a machine
CNAME	Specify an alias for a name
HINFO	Information about a machine
MB	Mailbox at which a user receives mail
MG	Membership of a mailing list
MINFO	Mailing list maintenance information
MR	Mail alias for a user
MX	Accept mail for another host
NS	Name server for a domain
PTR	Pointer to another location in the domain

SOA	Start of authority for a zone
TXT	Text data
WKS	Services available at an address through a given protocol

data The data portion of a resource record depends on the record type. If enclosed by parentheses, data may span more than one line; otherwise, the end of the line is taken as the end of the resource record. For a complete explanation of the data formats for the various resource records, and their use, see *UNICOS Networking Facilities Administrator's Guide*, Cray Research publication SG–2304.

EXAMPLES

The following is an example master file that describes a zone that contains two hosts:

```
$ORIGIN          ourdomain.com

@           IN   SOA  host.ourdomain.com.  admin.host.ourdomain.com. (
                 1        ; Serial
                 3600     ; Refresh
                 300      ; Retry
                 3600000  ; Expire
                 3600     ; Mininum
                 )
            IN   NS       host.ourdomain.com.
            IN   MX       0 host.ourdomain.com.
            IN   A        123.45.67.89

host        IN   A        123.45.67.89
            IN   HINFO    Cray-2S/4-128 UNICOS
            IN   WKS      123.45.67.89 TCP ( Telnet FTP )
cray        IN   CNAME    host

station     IN   A        123.45.67.90
            IN   HINFO    Generic Co. WS-1 UNIX
            IN   WKS      123.45.67.90 TCP ( Telnet FTP )
            IN   WKS      123.45.67.90 UDB ( Who )
```

The following is a master file that maps, in reverse, the addresses in the previous master file:

```
$ORIGIN          123.IN-ADDR.ARPA

@           IN  SOA  host.ourdomain.com.  admin.host.ourdomain.com. (
                1        ; Serial
                3600     ; Refresh
                300      ; Retry
                3600000  ; Expire
                3600     ; Mininum
                )
            IN  NS       host.ourdomain.com.

45.67.89    IN  PTR      host.ourdomain.com.
45.67.90    IN  PTR      station.ourdomain.com.
```

FILES

`/etc/named.boot` Domain name server configuration file

SEE ALSO

`named.boot`(5)

`named`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

UNICOS Networking Facilities Administrator's Guide, Cray Research publication SG–2304

NAME

mib.txt, snmpd.defs – Management information base for SNMP applications and SNMP agents, respectively

SYNOPSIS

/etc/mib.txt

/etc/snmpd.defs

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The mib.txt and snmpd.defs files define the management information base that the simple network management protocol (SNMP) uses. For a detailed explanation of the contents of this file, see RFCs 1155, 1212, and 1213. The mib.txt file defines the management information for SNMP applications such as snmpwalk. You can change this file to add additional information that agents support in other machines on the network.

Agent snmpd gets this information from the /etc/snmpd.defs file. This file is a compiled version of the management information base (MIB) that describes only the variables the agent supports. You should not change the snmpd.defs file.

FILES

/etc/mib.txt	File that defines MIB for SNMP applications
/etc/snmpd.defs	File that defines MIB for SNMP agents

SEE ALSO

snmpd(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

RFCs 1155, 1212, 1213

NAME

mnttab – Mounted file system table format

SYNOPSIS

```
#include <mnttab.h>
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The /etc/mnttab file contains a table of devices that are mounted by the mount(8) command. The mnttab file has the following structure, as defined by the mnttab.h include file:

```
struct  mnttab
{
        char    mt_dev[128],
                mt_filsys[128];
        short   mt_ro_flg;
        time_t  mt_time;
        char    mt_fstyp[16]  ;
        char    mt_mntopts[128]  ;
} ;
```

The fields in the mnttab structure have the following meanings:

mt_dev — Null-padded name of the directory on which the device is mounted (the *mount point*).

mt_filsys — Null-padded root name of the mounted special file. For more information on file system description files, see *General UNICOS System Administration*, Cray Research publication SG–2301.

mt_ro_flg — Mounted device's read and write permissions.

mt_time — Date the device was mounted.

mt_fstyp — Null-padded string that specifies the file system type. The file system type can be one of the following:

- NC1FS — UNICOS file system on Cray PVP systems
- SFS — UNICOS shared file system
- NFS — UNICOS NFS file system
- PROC — /proc file system
- INODE — /inode file system

For more information, see fstab(5), proc(4), and inode(4).

`mt_mntopts` Array that contains the text of the mount options specified after the file system type. For example, the file system specification `NFS,timeo=7` places the value `timeo=7` in `mt_mntopts`. This field is used only if the NFS file system types is specified. For a description of the options available, see `mount`(8).

The maximum number of entries in `mnttab` is based on the `NMOUNT` system parameter, which is located in the `config.h` include file; `NMOUNT` defines the number of mounted devices allowed.

FILES

`/etc/mnttab`	Mounted device table
`/usr/include/mnttab.h`	Structure of entries in `/etc/mnttab`

SEE ALSO

`fstab`(5), `proc`(4)

`mknod`(8), `mount`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

General UNICOS System Administration, Cray Research publication SG–2301

NAME

motd – File that contains message of the day

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The /etc/motd file is an ASCII file that contains the message of the day (MOTD). Its contents are displayed by /etc/profile or /etc/cshrc (see profile(5) or cshrc(5), respectively). This occurs at the start of an interactive or batch session, before the execution of the .profile file (for standard shell users) or the .login and .cshrc files (for C shell users).

Super users can create and modify the /etc/motd file. By convention, it contains short messages of interest to all users. A common motd file contains the machine type and operating system version, the system name, mention of scheduled down time, and announcements of software availability.

FILES

/etc/motd Message of the day file

SEE ALSO

cshrc(5), issue(5), profile(5)

login(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

NAME

msg – Message queue structures

SYNOPSIS

```
#include <sys/msg.h>
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The msg man page describes the constant and members of the structure msqid_ds in the sys/msg.h include file.

The following data types are defined through typedef:

msgqnum_t Used for the number of messages in the message queue.

msglen_t Used for the number of bytes allowed in a message queue.

These types are unsigned integer types that can store values at least as large as a unsigned short type.

The following message operation flag can be specified:

```
MSG_NOERROR
```

If this flag is specified and then a message that is larger than the buffer specified is received, the message is truncated and an error is not received.

The msqid_ds structure contains the following members:

```
struct msqid_ds {
        struct ipc_perm msg_perm;       /* operation permission struct */
        struct msg      *msg_first;     /* ptr to first message on q */
        struct msg      *msg_last;      /* ptr to last message on q */
        msglen_t        msg_cbytes;     /* current # bytes on q */
        msgqnum_t       msg_qnum;       /* # of messages on q */
        msglen_t        msg_qbytes;     /* max # of bytes on q */
        pid_t           msg_lspid;      /* pid of last msgsnd */
        pid_t           msg_lrpid;      /* pid of last msgrcv */
        time_t          msg_stime;      /* last msgsnd time */
        long            msg_pad1;       /* reserved for time_t expansion */
        time_t          msg_rtime;      /* last msgrcv time */
        long            msg_pad2;       /* time_t expansion */
        time_t          msg_ctime;      /* last change time */
        long            msg_pad3;       /* time expansion */
        long            msg_pad4[4];    /* reserve area */
};
```

The msgsnd(2) and msgrcv(2) system calls are used to send and receive the messages, respectively. The pid_t, time_t, key_t, and size_t types are defined as described in the sys/types.h file.

The following are declared as functions and also may be defined as macros:

```
int msgctl (int msqid, int cmd, struct msqid_ds *buf);
int msgget (key_t key, int msgflag);
int msgrcv (int msqid, void *msgp, size_t msgsz,
            long int msgtyp, int msgflg);
int msgsnd (int msqid, const void *msgp, size_t msgsz, int msgflg);
```

When this header file is included, all of the symbols from the sys/ipc.h file also will be defined.

SEE ALSO

ipc(5), types(5)

msgctl(2), msgget(2), msgrcv(2), msgsnd(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

ipc(7) Online only

NAME

named.boot – Domain name server configuration file

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The /etc/named.boot file contains initial configuration information for the named domain name server. The file lists administrative zones for which the local server has authority and the location (either a master file or another server) at which the local server will find the authoritative data for each zone.

A named.boot file is a text file that contains lines that consist of a keyword and one or more fields separated by spaces. Legal keyword lines and their formats are as follows:

```
directory    directory_name
cache        domain_name          masterfile
primary      domain_name          masterfile
secondary    domain_name          Internet_address [...] file
forwarders   Internet_address [...]
; slave
```

The *masterfile* argument is the name of a text file that contains authoritative data for the associated zone. The *masterfile* is specified in the Internet standard master file format. For the format of this file, see masterfile(5). Any line that begins with a ; symbol is a comment line and is ignored; blank lines also are ignored.

The directory line causes the server to change its working directory to the directory specified. This capability can be important for the correct processing of $INCLUDE files in primary zone files.

The cache line specifies that data in the specified master file will be placed in the back-up cache. Its primary use is to specify data such as locations of root domain servers. This cache is not used during typical operation, but it is used as an aid to find the current root servers. You can specify more than one cache file. You should retrieve the master file for the Internet root servers periodically from FTP.RS.INTERNIC.NET, because the list of root servers changes.

The primary line states that the specified master file contains authoritative data for the specified zone. The specified file is a master copy of the domain name system (DNS) data for the zone (that is, this host is a primary name server for the zone).

The secondary line states that the information for the specified zone will be transferred from a primary name server at the specified Internet address and saved in the specified back-up file. The host at that Internet address should be a primary name server for the zone. You should specify several primary nameserver addresses for this zone so that at least one is always reachable. When named starts, the zone information is loaded from the back-up file. When named receives a zone update, the back-up file is updated automatically.

The `forwarders` line specifies the addresses of other name servers that accept recursive queries from the local `named`. If the boot file specifies one or more forwarders, `named` sends all queries for data not in the cache to the forwarders first. Each forwarder is then asked, in turn, until an answer is returned or the list is exhausted. If no answer is forthcoming from a forwarder, `named` continues as it would have without the forwarders line, unless it is in `forward-only` mode. The forwarding facility is useful to cause a large, sitewide cache to be generated on a master, and to reduce traffic over links to outside servers. You also can use the forwarding facility to allow name servers to run that do not have access directly to the Internet, but want to perform as though they do have access.

The `slave` directive (not shown) is allowed for backward compatibility. Its meaning is identical to `options forward-only`.

You can use the `xfrnets` directive (not shown) to implement primitive access control. If this directive is given, your name server answers only zone transfer requests from hosts that are on networks listed in your `xfrnets` directives. This directive also may be given as `tcplist` for compatibility with older, interim servers.

You can use the `include` directive (not shown) to process the contents of some other file as though they appeared in place of the `include` directive. This capability is useful if you have numerous zones or if you have logical groupings of zones that various people maintain. The `include` directive accepts one argument: the name of the file with the contents that will be included. Quotation marks are not necessary around the file name.

The `bogusns` directive (not shown) tells `named` that no queries will be sent to the specified name server addresses, which are specified as dotted quads, not as domain names. This capability is useful when you know that some popular name server has bad data in a zone or cache, and you want to avoid contamination while the problem is being fixed.

The `limit` directive can be used to change BIND's internal limits, some of which (e.g., datasize) are implemented by the system and others (e.g., `transfers-in`) by BIND itself. The number following the limit name can be scaled with "k" (kilobytes), "m" (megabytes), or "g" (gigabytes). The currently defined arguments are as follows:

`datasize` This sets the process data size enforced by the kernel. Not all systems provide a call to implement this argument. Use of the `datasize` parameter on systems without this call will result in a warning message.

`transfers-in`
This sets the number of `named-xfer` subprocesses which BIND will spawn at any one time. `transfers-per-ns` This defines the maximum number of zone transfers to be simultaneously initiated to any given remote name server.

The `options` directive introduces a boolean specifier that changes the behavior of BIND. More than one option can be specified in a single directive. The currently defined options are as follows:

`no-recursion`
: This option will cause BIND to answer with a referral rather than actual data whenever it receives a query for a name it is not authoritative for; this option should not be set on a server that is listed in any host's `resolv.conf` file.

`query-log`
: This option causes all queries to be logged via `syslog`(3). This option should be used with caution; the log uses a large amount of disk space.

`forward-only`
: This option causes the server to query only its forwarders. This option is normally used on a machine that wishes to run a server, but for physical or administrative reasons, cannot be given access to the Internet.

`fake-iquery`
: This option instructs BIND to send back a useless and bogus reply to `inverse queries` rather than responding with an error. This option is helpful for sites with multiple microcomuters, SunOS hosts, or both.

You can use the `max-fetch` directive (not shown) to override the default limit (10) to the number of `named-xfer` subprocesses that `named` can spawn at any one time.

For a complete description of each keyword line and other information, see Appendix D in the *UNICOS Networking Facilities Administrator's Guide*, Cray Research publication SG–2304.

NOTES

The boot file directives `domain` and `suffixes` are obsolete with the introduction of a more useful resolver-based implementation to add suffixes to partially qualified domain names. The prior mechanisms could fail under certain situations, especially when the local name server did not have complete information.

EXAMPLES

An example of a `/etc/named.boot` file follows:

```
directory  /usr/domains

; ------------------------------------------------------
; name     zone                          file

cache       .                            cache

primary    0.0.127.IN-ADDR.ARPA          local

primary    ourdomain.com                 ourdomain
primary    54.321.IN-ADDR.ARPA           ourdomain.rev

; ------------------------------------------------------
; name     zone             addresses    file

secondary  theirdomain.edu  34.56.78.90 56.78.90.12  theirdomain
secondary  34.IN-ADDR.ARPA  34.56.78.90 56.78.90.12  theirdomain.rev
```

FILES

`/etc/named.boot`	Contains initial configuration information for the domain name server

SEE ALSO

`masterfile`(5)

`named`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

UNICOS Networking Facilities Administrator's Guide, Cray Research publication SG–2304

NAME

netgroup – List of network groups

SYNOPSIS

/etc/netgroup

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The netgroup file defines networkwide groups used for permission checking when doing remote mounts, remote logins, and remote shells. For remote mounts, the information in netgroup is used to classify machines; for remote logins and remote shells, it is used to classify users. Each line of the netgroup file defines a group and has the format *groupname members*; *members* is either another group name or a triple of the format (*hostname*, *username*, *domainname*). Any of these fields can be empty, in which case, the empty field signifies a wildcard. Thus, universal (,,) defines a group to which everyone belongs. (In this case, all three fields are wildcards).

The *domainname* field must be either the local domain name or empty for the network group entry to be used. This field does not limit the network group or provide security. The *domainname* field refers to the domain in which the triple is valid, it does not refer to the domain that contains the trusted host.

A gateway machine must be listed under all possible host names by which it can be recognized, as in the following example:

```
wan (gateway,,) (gateway-ebb,,)
```

Field names that begin with something other than a letter, digit, or underscore (such as -) work in the opposite fashion. For example, consider the following entries:

```
justmachines  (analytica,-,sun)
justpeople  (-,babbage,sun)
```

Machine analytica belongs to group justmachines in domain sun, but no users belong to it. Similarly, user babbage belongs to group justpeople in domain sun, but no machines belong to it.

NOTES

The netgroups feature port was designed to work only with the network information system (NIS). NIS must be configured and running on the system to implement netgroups.

WARNINGS

The triple (,,*domainname*) allows all users and machines trusted access, and it has the same effect as the triple (,,,).

To restrict access to a specific set of members correctly, use the *hostname* and *username* fields of the triple.

SEE ALSO

`exports`(5)

`makedbm`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

netrc – TCP/IP autologin information file for outbound ftp(1B) requests

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The .netrc file contains login information required for ftp(1B) access to a remote machine. When ftp(1B) is opening a connection to a specified remote machine, it checks for this file in the user's home directory on the machine initiating the file transfer. If the file exists, ftp(1B) checks it for login information for the specified machine.

The .netrc file contains one or more entries; each entry describes default values and macros to use when connecting to a specified remote host. Each entry contains token pairs that consist of a key word and a value. Five key words are recognized: machine, login, password, account, and macdef.

The machine *remote_hostname* token pair defines the start of an entry; all other token pairs are optional and may be given in any order, though they usually are given in the order that follows:

```
machine   remote_hostname
login     login_name
password  password
account   account_name
macdef    macro_name macro
```

Usually, each entry is on one line. Each token is a string of characters, separated by a space, tab, comma, or newline character, or a string of characters between two double quotation marks. The \ symbol is a special character, and you can embed any of the special characters (space, tab, comma, newline, double quotation marks, and backslash) into a token by preceding it with a backslash. The macdef token pair is different from the other token pairs, in that after the macdef *macro_name* token pair, all characters up to a blank line are assumed to be the definition of the macro.

As a security precaution, ftp(1B) requires that .netrc be readable and writable only by its owner if it contains any password or account fields in any of the entries. If .netrc does not exist, or if it exists but contains no entry or a partial entry for the specified machine, the user is prompted for the missing information as needed.

CAUTIONS

Use of passwords in the .netrc file creates a major security hole in the network. For security reasons, passwords should not appear in files, even protected ones. Use the .netrc file without the password entries.

EXAMPLES

An example of a .netrc file follows:

```
machine biology login bonnie
machine chemistry login bonnie2
        macdef lsf
        ls -CF

        macdef pwdlsf
        pwd
        ls -CF

machine blackhole login anonymous password bonnie
```

FILES

$HOME/.netrc	Contains login information required for ftp(1B) access to a remote machine

SEE ALSO

hosts(5)

ftp(1B) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

rexec(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

NAME

networks – Network name database

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The /etc/networks file contains information regarding the known networks that compose the Internet and network. For each network, a line contains the network type, the official network name, the network number or address, and any aliases that exist for the network name. Items are separated by any number of blanks, tab characters, or a combination of the two. A # symbol indicates the beginning of a comment; additional characters up to the end of the line are not interpreted by the routines that search the file.

The supported network type is inet for Internet network entries. If you do not specify a network type, inet is assumed.

For hosts on the DARPA Internet, this file may be created from the official network database maintained at the Network Information Center (NIC), though local changes may be required to bring it up-to-date regarding unofficial aliases or unknown networks.

You can specify network numbers in the conventional "." (dot) notation by using the inet_network routine from the Internet address manipulation library, inet(3C).

Network names may contain any printable character other than a field delimiter, newline character, or comment character.

EXAMPLES

The following is an example from an /etc/networks file:

```
#
#   Internet networks
#
loopback  127
crayhy    84
backbone  192.9.0
nobelnet  192.9.1
arsonnet  192.9.3
pubsnet   192.9.30
inet      softnet    192.6.2
```

FILES

`/etc/networks` Contains network information

SEE ALSO

`getnet`(3C), `inet`(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

NAME

nl_types – Defines message system variables

SYNOPSIS

```
#include <nl_types.h>
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The <nl_types.h> header file is required for C programs that use the UNICOS message system. This file defines various types, macros, and functions that the message system uses.

The nl_catd type definition is defined as the type of message catalog file descriptors.

The NL_MSGSET macro is defined as the set number for all messages, and the NL_EXPSET macro is defined as the set number for all explanations. These macros expand to integral constant expressions that have distinct values, suitable for use as the second argument to the catgetmsg(3C) and catgets(3C) functions.

The NL_CAT_LOCALE macro is defined as the *oflag* argument to the catopen(3C) function that causes the LC_MESSAGES category to be used instead of the LANG environment variable.

The <nl_types.h> header file declares the following functions, which reside in the /lib/libc.a file. For complete descriptions of these functions, see the man pages.

Function	Description
catopen(3C)	Opens message catalog
catclose(3C)	Closes message catalog
catgetmsg(3C)	Retrieves message to user buffer
catgets(3C)	Retrieves pointer to message
catmsgfmt(3C)	Formats message for printing

These routines use the NLSPATH, LANG, and MSG_FORMAT user environment variables and the LC_MESSAGES category in their processing. The NLSPATH and LANG variables are described on the catopen(3C) man page, and the MSG_FORMAT variable is described on the explain(1) man page. The LC_MESSAGES category is described on the locale(1) man page.

SEE ALSO

`caterr`(1), `catxt`(1), `explain`(1), `gencat`(1), `whichcat`(1) describe message system user commands
`locale`(1) describes the `LC_MESSAGES` category
in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

`catgetmsg`(3C), `catgets`(3C), `catmsgfmt`(3C), `catopen`(3C) describe message system library functions in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

Cray Message System Programmer's Guide, Cray Research publication SG–2121, contains details about all aspects of the message system

NAME

passwd – Format of the password file

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The /etc/passwd file is an ASCII file that contains one entry for each user on the system. udbgen(8) maintains this file automatically for compatibility with older system versions, but the system does not use it for user validation.

Each entry contains the following fields:

- Login name
- Encrypted password
- Numeric user ID (UID)
- Numeric group ID (GID)
- Comment
- Initial working directory
- Shell (program to use as shell)

Each field in the user entry is separated from the next by a colon. The comment field can contain any desired information; however, it cannot contain the colon or newline characters. Each user entry is separated from the next by a newline character. The password field is present for compatibility, but it is always set to *. If password aging is active, the * will be followed by a comma and the age control string (see libudb(3C)).

FILES

/etc/passwd	Password file
/etc/udb	User database file
/etc/udb_2/udb.index	Public extension index file
/etc/udb_2/udb.priva	Private field extension file
/etc/udb_2/udb.pubva	Public field extension file
/etc/udb.public	Public user database file

SEE ALSO

acid(5), group(5), udb(5)

libudb(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

udbgen(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

printcap – Printer capability database

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The /etc/printcap file is used when you send output to a printer by using the lpr(1B) command. The /etc/printcap file contains a list of names by which the lpr(1B), lpq(1B), and lprm(1B) commands know each printer. These printers are not necessarily connected directly to the system, but they are available anywhere in the TCP/IP network. For a discussion of how to set up the database for a given printer, see *UNICOS Networking Facilities Administrator's Guide*, Cray Research publication SG–2304.

Each entry in the database describes one printer. The spooling system accesses the printcap file each time a print command is used. This allows dynamic addition and deletion of printers.

The default printer is usually lp; to change the default printer, use the #PRINTER environment variable. Each spooling utility supports a -P*printer* option to specify a destination printer other than the default.

Each entry in the printcap file describes a printer; it is a line that consists of a number of fields separated by : symbols. The first entry for each printer gives the names that are known for the printer, separated by | symbols. You should not use blanks in the printer's name because this requires users to enclose the name in quotation marks when specifying the -P option on the lpr(1B) command. Entries can continue onto multiple lines through the use of a \ as the last character of a line, and empty fields may be included for readability.

Printer capabilities, all introduced by 2-character codes, are of three types: Boolean, numeric, and string. These capabilities are shown in the table that follows the description of the capabilities.

Boolean capabilities indicate that the printer has a particular feature. Boolean capabilities are indicated by the word bool in the Type column of the capabilities table that follows. Their presence in the printcap file indicates that the associated feature is present on the printer being described.

Numeric capabilities supply information such as baud rates, number of lines per page, and so on. Numeric capabilities are indicated by the word num in the Type column of the capabilities table that follows. Numeric capabilities are entered as the 2-character capability code, followed by the # symbol, followed by the numeric value (for example, :br#1200: is a numeric entry that states that this printer should run at 1200 Bd).

String capabilities provide a device name, file name, or character sequence that can be used to perform a particular printer operation such as cursor motion. String capabilities are indicated by the word str in the Type column of the capabilities table that follows. String capabilities are entered as the 2-character capability code followed by an = symbol, and then a string ending at the next following : (for example, :rp=spinwriter: is a string entry states that the remote printer is named spinwriter).

The following table shows printer capabilities:

Name	Type	Default	Description
af	str	Null	Specifies name of the accounting file. This capability is useful only for locally attached printers.
br	num	None	Sets the baud rate (ioctl request) if lp is a tty.
cf	str	Null	Specifies data filter for cifplot.
df	str	Null	Specifies TeX data filter (DVI format).
du	str	0	Specifies user ID of user daemon.
fc	num	0	Clears flag bits (sgtty.h) if lp is a tty.
ff	str	\f	Sends this string for a form feed.
fo	bool	False	Prints a form feed when device is opened.
fs	num	0	Sets flag bits (sgtty.h).
gf	str	Null	Graphs data filter (plot(3) format).
hl	bool	False	Prints the burst header page last.
if	str	Null	Specifies name of text filter that does accounting. This capability is useful only for locally attached printers.
lf	str	/dev/console	Specifies error logging file name.
lo	str	Lock	Specifies name of lock file.
lp	str	/dev/lp	Specifies device name to open for output.
ma	str	level0,077	Specifies maximum security label allowed.
mc	num	0	Specifies maximum number of copies.
mi	str	level0,0	Specifies minimum security label allowed.
mx	num	1000	Specifies maximum file size (in BUFSIZ blocks), 0 = unlimited.
nf	str	Null	Specifies ditroff (device-independent troff) data filter.
of	str	Null	Specifies name of output filtering program. This capability is useful only for locally attached printers.
pl	num	66	Specifies page length (in lines). This capability is useful only for locally attached printers.
pw	num	132	Specifies page width (in characters). This capability is useful only for locally attached printers.
px	num	0	Specifies horizontal page width (in pixels).
py	num	0	Specifies vertical page length (in pixels).
rf	str	Null	Specifies filter for printing Fortran-style text files.
rg	str	Null	Specifies restricted group (only group members are allowed access).
rm	str	Null	Specifies machine name for remote printer.
rp	str	lp	Specifies remote printer name argument.
rs	bool	False	Restricts remote users to those with local accounts.
rw	bool	False	Opens printer device read/write instead of read-only.
sb	bool	False	Specifies short banner (one line only).

Name	Type	Default	Description
sc	bool	False	Suppresses multiple copies.
sd	str	/usr/spool/lpd	Specifies spool directory.
sf	bool	False	Suppresses form feeds.
sh	bool	False	Suppresses printing of burst page header.
st	str	Status	Specifies status file name.
tf	str	Null	Specifies troff data filter.
tr	str	Null	Specifies trailer string to print when queue empties.
vf	str	Null	Specifies raster image filter.
xc	num	0	Clears local mode bits, if lp is a tty.
xs	num	0	Sets local mode bits.

NOTES

Error messages sent to the console have a carriage return and a line feed appended to them, rather than just a line feed.

If the local line printer driver supports indentation, the daemon must understand how to invoke it.

The fs, fc, xs, and xc fields are flag *masks*, rather than flag *values*. Certain default device flags are set when the device is opened by the line printer daemon if the device is a tty device. The flags indicated in the fc field are then cleared; the flags in the fs field are then set (or vice versa, depending on the order of fc#*nnnn* and fs#*nnnn* in the /etc/printcap file). For example, to set the flags 06300 in the fs field, enter the following:

```
:fc#0177777:fs#06300:
```

The same process applies to the xc and xs fields.

Two output filtering programs are supplied with Cray Research software. They are installed in the /usr/lib directory when lpr is installed. These programs process print files as follows:

lpf Reads nroff output and converts lines that begin with ^H to overwritten lines. It also handles multiple overwritten lines.

necf Reads the file to be printed and writes it to the printer one line at a time, as installed by default. This filter program is only a skeleton program as supplied by Cray Research. If TTY is defined during compile time, this program changes newline characters to carriage return characters, followed by line-feed characters. If SHEETFEEDER is defined during compile time, this program adds page ejects after every 66 lines to the file being printed.

EXAMPLES

The following is an example of a simple `printcap` file. Three printers are defined in this example. The first line of the file is a comment.

The second line lists information for a printer named `ps0`. Printer `ps0` is connected to the remote system `nobel`, rather than existing on the local system. All files to be printed on this printer are sent to `nobel` and are printed on the `nobel` printer named `nobel_0`. All files are spooled to the `/usr/spool/ps0` directory before being sent to the remote system for printing.

The last two lines define the other two printers in a similar manner. The second printer has specified `lp=`, which indicates that no local `/dev` entry exists for this printer. If you omit `lp=`, it defaults to `/dev/lp`. When you specify `rm=remote_name`, the `lp` parameter is ignored.

```
#     Sample printcap file
ps0:rm=nobel:rp=nobel_0:sd=/usr/spool/ps0:
ps1:lp=:rm=fermi:rp=fermi_10:sd=/usr/spool/ps1:
ps2:rm=sobrero:rp=sobrero_5:sd=/usr/spool/ps2:
```

SEE ALSO

`lpq`(1B), `lpr`(1B), `lprm`(1B) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

NAME

profile – Format of shell start-up file

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The /etc/profile and $HOME/.profile files are shell start-up files. On login, the system checks the shell field in a user's entry in the UDB file (/etc/udb) to see what shell it specifies. If you specify /bin/sh or /bin/ksh, /etc/profile then $HOME/.profile is run. If you specify /bin/csh, the C shell environment is run. For more information on the C shell, see csh(1) and cshrc(5).

If /bin/sh or /bin/ksh is specified in the shell field of the password file, the following actions occur as a user logs in:

1. If the /etc/profile file exists, the POSIX shell (sh(1)) or Korn shell (ksh(1)) executes it. Among other operations, /etc/profile prints /etc/motd, the message of the day if that file exists (see motd(5)).
2. If the user's login directory contains a file named .profile, sh(1) or ksh(1) executes it. For the Korn shell (ksh(1)), if the ENV environment variable is set, parameter substitution is performed on the value. The result is expected to be a path name of a script that ksh(1) executes.
3. The user's terminal session begins.

The .profile file is useful for setting exported environment variables. (The environment variable for the time zone, TZ, is set in the /etc/inittab file. For more information, see inittab(5).)

EXAMPLES

An example of a typical .profile file follows:

```
#  Make some environment variables global
export MAIL PATH
#  Set file creation mask
umask 22
#  Tell me when new mail comes in
MAIL=/usr/mail/myname
#  Add my /bin directory to the shell search sequence
PATH=$PATH:$HOME/bin
```

FILES

`$HOME/.profile`	Shell start-up file in user's home directory
`/etc/profile`	Systemwide shell start-up file

SEE ALSO

`cshrc`(5), `inittab`(5), `motd`(5), `term`(5), `udb`(5)

`csh`(1), `env`(1), `ksh`(1), `login`(1), `mail`(1), `stty`(1), `su`(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

NAME

proto – Prototype job file for at(1)

SYNOPSIS

/usr/lib/cron/.proto

/usr/lib/cron/.proto.*queue*

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

When a job is submitted to at(1) or batch(1), the job is constructed as a shell script. The shell script is constructed by a set of standard shell commands that makes the environment (see environ(7)) for the at(1) job the same as the current environment. The at(1) utility then reads a prototype file, appending the contents after the environment. Last, the commands specified as input to the at(1) command are appended after the prototype file commands.

Text from the prototype file is copied to the job file, except for special variables that are replaced by other text:

Variable	Replaced by
$a	Account ID of the user
$d	Current working directory
$l	Current file size limit (see ulimit(2))
$m	Current umask (see umask(2))
$t	Time at which the job should be run, expressed as seconds since January 1, 1970, 00:00 Greenwich Mean Time, preceded by a colon
$<	Text read by at(1) from standard input (that is, the commands provided to at(1) to be run in the job)

When the job is submitted in queue *queue*, at(1) uses the /usr/lib/cron/.proto.*queue* file as the prototype file if it exists; otherwise, it uses the /usr/lib/cron/.proto file.

EXAMPLES

The standard .proto file supplied with the UNICOS operating system is as follows:

```
# USMID @(#)man/man5/proto.5  100.0 07/15/97 14:39:30
newacct $a
cd $d
ulimit $l
umask $m
unset TMPDIR
export TMPDIR
$<
```

This file creates commands that change the account to the current user, change the current directory in the job to the current directory at the time `at`(1) was run, set the file size limit to the current file size allowed on the system, and change the umask in the job to the umask at the time `at`(1) was run, to be inserted before the commands in the job. The `TMPDIR` shell variable also is unset, because this value defines a temporary directory, which might not exist when the `at`(1) job is executed. Last, the commands provided to `at`(1) are appended after the `export` command.

FILES

`/usr/lib/cron/.proto`	Default prototype file for `at`(1) or `batch`(1) job
`/usr/lib/cron/.proto.`*queue*	Prototype file for `at`(1) or `batch`(1) job submitted in queue *queue*

SEE ALSO

`at`(1), `batch`(1), `ksh`(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

`ulimit`(2), `umask`(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

`environ`(7) (available only online)

NAME

protocols – Protocol name database

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The /etc/protocols file contains information about the known protocols used in the Internet network. For each protocol, one line should contain the official protocol name, the protocol number, and any aliases that exist for the protocol name. Items are separated by any number of blanks and/or tab characters. A # symbol indicates the beginning of a comment; if you specify this character, routines that search the file do not interpret additional characters up to the end of the line.

Protocol names may contain any printable character other than a blank, tab, newline, or comment (#).

EXAMPLES

The following example shows typical entries in /etc/protocols:

```
# Internet (IP) protocols

ip    0  IP    #  internet protocol, pseudo protocol number
icmp  1  ICMP  #  internet control message protocol
tcp   6  TCP   #  transmission control protocol
udp   17 UDP   #  user datagram protocol
```

FILES

/etc/protocols	Contains information about known protocols

SEE ALSO

getprot(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

NAME

publickey – Public key database

SYNOPSIS

/etc/publickey

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The public key database, /etc/publickey, is used for secure networking. Each entry in the database consists of a network user name (which may refer either to a user or a host name), followed by the user's public key (in hexadecimal notation), a colon, and then the user's secret key encrypted with its login password (also in hexadecimal notation).

This file is altered either by the user through the chkey(1) command or by the system administrator through the newkey(8) command. The /etc/publickey file should contain only data on the network information service (NIS) master machine, where it is converted into the NIS database publickey.byname. Cray Research strongly recommends that the Cray Research system not be used as the NIS master machine for any NIS database, including publickey.

SEE ALSO

chkey(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

publickey(3R) in the *Remote Procedure Call (RPC) Reference Manual*, Cray Research publication SR–2089

newkey(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

UNICOS Networking Facilities Administrator's Guide, Cray Research publication SG–2304

NAME

queuedefs – Queue description file for at(1), batch(1), and cron(8)

SYNOPSIS

/usr/lib/cron/queuedefs

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The queuedefs file maintains definitions for all queues that cron(8) manages. If this file does not exist, the default values are used. Each noncomment line of queuedefs describes one queue and must have the following format:

*q.NN*j*NN*n*NN*w

q The name of the job queue; must be a letter, a through z. a is the default queue for jobs that at(1) starts. b is the default queue for jobs that batch(1) (see at(1)) starts.

*NN*j The limit on jobs that can run at any one time for the job queue. *NN* is an integer; the default is 100.
NOTE: You can increase the value *NN*j only up to the value of MAXRUN in cron(8). The default value of MAXRUN is 25. MAXRUN can be increased by using the -m option to cron(8).

*NN*n The nice(1) value assigned to each command executed for the job queue. *NN* is an integer; the default is 2.

*NN*w The time (in seconds) that cron(8) waits before reexecuting a command, if the command could not run at the first execution because all criteria for execution were not met. The default is 60.

Empty fields are initialized to the default values.

Lines that begin with # are comments and are ignored.

EXAMPLES

The following is an example of a queuedefs file:

```
a.4jln
b.2j2n90w
```

This file specifies that the a queue, for at(1) jobs, can have up to four jobs running simultaneously; those jobs will be run with a nice(1) value of 1. Because a w (wait) value was not given, if a job cannot be run because too many other jobs are running, cron(8) will wait 60 seconds before trying again to run it. The b queue, for batch(1) jobs, can have no more than two jobs running simultaneously; those jobs will be run with a nice(1) value of 2. If a job cannot be run because too many other jobs are running, cron(8) will wait 90 seconds before trying again to run it.

All other queues can have up to 100 jobs running simultaneously; they will be run with a nice(1) value of 2, and if a job cannot be run because too many other jobs are running, cron(8) will wait 60 seconds before trying again to run it.

Changes to queue definitions take effect before the cron(8) daemon executes the next job.

FILES

/usr/lib/cron/queuedefs Defines all queues managed by cron(8)

SEE ALSO

at(1), nice(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

cron(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

quota – Quota control file format

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

File quota enforcement controls the amount of file system space consumed by placing an upper bound on the number and size of files available to an account, group, or user. All quota control information resides in quota control files.

One quota control file exists per controlled file system or a number of file systems organized into a *quota control group*. The default name for the quota control file is `.Quota60`. By default, when a quota control file is associated with one file system, the file appears in the root directory of its related file system. In the case of quota control groups, you must specify explicitly the location and name of the file. The `quadmin`(8) command allows the name and location of a quota control file to be other than the default.

The system administrator creates and modifies quota control files by using the `quadmin`(8) command. For more information about all aspects of administering file quotas, see the description of the command in *UNICOS Resource Administration*, Cray Research publication SG–2302.

The quota control file consists of one header structure followed by the hash table with the remainder of the file consisting of an arbitrary number of quota control structures. The file has a hashed access organization that use the ID value as the key. Quota control structures that have IDs with synonymous hash values are linked together with the most recently added record at the head of the chain.

IDs can belong to any one of three control classes. The control classes are account IDs (`acid`), group IDs (`gid`), and user IDs (`uid`).

The header and control structures are defined in the `sys/quota.h` file.

Quota File Header

The `qf_header` structure identifies the quota file and contains general information needed by the various components of the file system quota control feature. The `qf_header` appears at offset 0 in the quota file and consists of 1256 bytes. The first object in the header is a magic number that is used to determine whether the file has been generated on a release of UNICOS that has compatible quota control files.

The qf_header structure is defined as follows:

```
struct qf_header {
       long      qf_magic;        /*  quota version identification       */
       struct    q_header
                 acct_h,          /* account header                      */
                 group_h,         /* group header                        */
                 user_h;          /* user header                         */
       time_t    qf_min_dm;       /* minimum data migration threshold    */
       uint      qf_lvl : 8,      /* Q_ON_DEFAULT enable level           */
                 qf_eval : 8,     /* evaluator selector                  */
                 qf_style : 1,    /* 1 if aggregate (DMF) quotas,        */
                                  /* 0 if online                         */
                 qf_spare : 47;   /* reserved                            */
       long      hef1,            /* field 1 reserved for evaluator's use */
                 hef2;            /* field 2 reserved for evaluator's use */
       uint      qf_qfnamesize;   /* size of qf_name[]                   */
       uint      qf_hashents;     /* length of the hash table in entries  */
       off_t     qf_hashtaboffs;  /* offset of the hash table            */
       char      qf_name[PATH_MAX+1];  /* name of the quota file         */
};
```

qf_magic
: Magic number to identify the format of the file; this field is reserved for future use with alternative file formats.

q_header
: Three q_header structures occur in the qf_header. Each of the control classes has a q_header structure: account, group, and user. The format of the q_header structure is defined following the description of the qf_header structure.

qf_min_dm
: Minimum data migration threshold. This field is reserved for future use.

qf_lvl
: Default enable level when Q_ON_DEFAULT is requested. This field records the quota enable level (*count*, *enforce*, or *inform*) that was last selected. The default on a newly created file is *count* (Q_ON_COUNT).

qf_eval
: Oversubscription evaluation algorithm selector. This may contain one of the values QEVAL_NONE, QEVAL_EXP, QEVAL_LIN, QEVAL_RSV1, or QEVAL_RSV2. If this field contains QEVAL_NONE, the default, oversubscription is disabled.

qf_style
: Flag for online or aggregate quotas. 0 means online; 1 means aggregate. If aggregate quotas are selected, both disk block online and then migrated offline by DMF are counted.

hef1
: Evaluation field 1. This field is reserved for use by the evaluation algorithm selected in qf_eval.

hef2
: Evaluation field 2. This field is reserved for use by the evaluation algorithm selected in qf_eval.

qf_qfnamesize
: Size (in bytes) of qf_name.

qf_hashents Number of *entries* in the hash table. The released system uses a hash table size of 2039.

qf_hashtaboffs
Byte offset from the beginning of the file to the hash table.

qf_name Name of the quota file. The kernel records the name most recently used to open the quota file in this field. quadmin(8) also uses this field for verification.

The q_header structure is defined as follows:

```
struct q_header {
        uint     hdr_flags;     /* header flags (QFC_HDR_xx)     */
        float    wf_fq;         /* file quota warning fraction   */
        float    wf_iq;         /* inode quota warning fraction  */
        uint     def_fq;        /* default file quota            */
        uint     def_iq;        /* default inode quota           */
        uint     warn_fq;       /* file quota warning value      */
        uint     warn_iq;       /* inode quota warning value     */
};
```

hdr_flags Header flags. This field sets the default quota enforcement mode. Valid modes are file quotas, inode quotas, or both.

wf_fq Warning fraction for file quota. A floating fraction in the range 0.0 < wf_fq < 1.0 that specifies where the default warning threshold will occur in relation to the file quota. Unmodified wf_fq structures contain a default value of 0.9, which places the warning window at 90% of the quota.

wf_iq Warning fraction for inode quota. A floating fraction in the range 0.0 < wf_iq < 1.0 that specifies where the default warning threshold will occur in relation to the inode quota. Unmodified wf_iq structures contain a default value of 0.9, which places the warning window at 90% of the quota.

def_iq Default inode quota. This field contains the inode quota that will be enforced if a quota entry indicates use of the default. Unmodified q_header structures contain a default value of 200 inodes for all control classes.

def_fq Default file quota. This field contains the file quota that will be enforced if a quota entry indicates use of the default. Unmodified q_header structures contain a default value of 5000 blocks for all control classes.

warn_fq Default file warning value in blocks. quadmin(8) computes this value based on def_fq and wf_fq. For example, if the def_fq file quota is 5000 and the wf_fq warning fraction is 0.9, this field will be set to 4500. When the amount of file space in use exceeds this value, a warning signal (SIGINFO) is issued.

`warn_iq` Default inode warning value. `quadmin`(8) computes this value based on `def_iq` and `wf_iq`. For example, if the `def_iq` inode quota is 200 and the `wf_iq` warning fraction is 0.9, this field will be set to 180. When the number of inodes in use exceeds this value, a warning signal (`SIGINFO`) is issued.

Quota Control Structures

Each ID, whether an account, group, or user ID, occupies one offset in the quota control file and has control information for each class of ID being controlled.

Each ID value created in the file consists of a `qf_entry` structure. This structure contains a quota control structure (`q_entry`) for each of the three ID classes (account, group, and user), along with identification and chaining fields.

Because all IDs of the same value in each ID class are not always defined, some of the space in each structure may be unused. For example, if your system has 123 defined as a user ID, but 123 does not occur as an account or group ID, only the `q_entry` structure named `user_q` will be occupied.

A `qf_entry` structure consists of 216 bytes. The location of the structure that corresponds to a specific ID is found by evaluating the following expression to access the correct hash table entry and following the chain rooted in that entry until the record is found:

```
(ID % 2039)
```

The format of the `qf_entry` entry is defined as follows:

```
struct   qf_entry   {
         struct     qf_ident
                    qf_ident;     /* record's identification      */
         uint       res1 : 32,    /* reserved space               */
                    id : 32;      /* id (account, group and user) */
         struct     q_entry
                    acct_q,       /* account quota                */
                    group_q,      /* group quota                  */
                    user_q;       /* user quota                   */
         off_t      q_next;       /* next record in hash chain    */
};
```

`qf_ident` Record's identification. The type and size of the record is kept in this structure for future multiple record type support.

`id` Account, group, or user ID to which the quota information pertains.

`q_entry` Account, group, and user quota control definitions. Each quota control structure is defined as follows:

```
struct q_entry  {
       time_t   f_wtime;          /* time when file warning was reached   */
       uint     f_quota : 32,     /* file quota (blocks)                  */
                f_runquota : 32;  /* running quota if non-zero            */
       uint     f_warn : 32,      /* file warning value                   */
                f_use : 32;       /* file usage (blocks)                  */
       uint     i_quota : 32,     /* inode quota (units)                  */
                res : 32;         /* reserved                             */
       uint     i_warn : 32,      /* inode warning value                  */
                i_use : 32;       /* inode usage (units)                  */
       uint     ef1 : 32,         /* field 1 reserved for evaluator's use */
                ef2 : 32;         /* field 2 reserved for evaluator's use */
       uint     ef3 : 32,         /* field 3 reserved for evaluator's use */
                ef4 : 32;         /* field 4 reserved for evaluator's use */
       long     ef5;              /* field 5 reserved for evaluator's use */
};
```

For the file and inode quota (`f_quota` and `i_quota`) and warning (`f_warn` and `i_warn`) special values have been defined. Except for 0 and `QFV_MINVALUE`, the special values are defined to be greater than the maximum value allowed in a field. The values are at the upper end of the range for 32-bit values and are defined in `sys/quota.h`. Their symbolic names are used here.

Value	Definition
`0`	Value not specified.
`QFV_MINVALUE`	Smallest value allowed in a field (1).
`QFV_MAXVALUE`	Largest value allowed in a field (4294967285)
`QFV_DEFAULT`	The quota value is determined by the corresponding default in the header.
`QFV_NOEVAL`	Quota control is disabled. The quota is unlimited.
`QFV_PREVENT`	No more resources may be allocated.

The fields in the `q_entry` structure are defined as follows:

Field	Definition
`f_wtime`	Time at which the first file warning was reached. Special values do not apply to this field.
`f_quota`	File quota or when oversubscription is enabled, the upper bound of file allocation. Special values apply to this field.
`f_runquota`	If this field is nonzero, this is the oversubscription threshold value. Special values apply to this field.
`f_warn`	File quota warning value. The special values mentioned previously apply to this field, except that `QFV_NOEVAL` means that a warning will never be issued.

`f_use` File usage. The current accumulated file usage. The kernel maintains this field during operation, and the `qudu`(8) command computes current file usage for initial quota setup or correction. Special values do not apply to this field.

`i_quota` Inode quota. The maximum number of inodes allowed for the ID. Special values apply to this field.

`i_warn` Inode quota warning value. The special values mentioned previously apply to this field, except that `QFV_NOEVAL` means that a warning will never be issued.

`i_use` Inode usage. The current number of inodes. The kernel maintains this field during operation and the `qudu`(8) command computes current inode usage for initial quota setup or correction. Special values do not apply to this field.

`ef1` Field reserved for evaluator's use. See the documentation on the specific algorithms in *UNICOS Resource Administration*, Cray Research publication SG–2302.

`ef2` Field reserved for evaluator's use. See the documentation on the specific algorithms in *UNICOS Resource Administration*, Cray Research publication SG–2302.

`ef3` Field reserved for evaluator's use. See the documentation on the specific algorithms in *UNICOS Resource Administration*, Cray Research publication SG–2302.

`ef4` Field reserved for evaluator's use. See the documentation on the specific algorithms in *UNICOS Resource Administration*, Cray Research publication SG–2302.

`ef5` Field reserved for evaluator's use. See the documentation on the specific algorithms in *UNICOS Resource Administration*, Cray Research publication SG–2302.

FILES

`sys/quota.h` Quota control definition file

SEE ALSO

`quadmin`(8), `qudu`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

UNICOS Resource Administration, Cray Research publication SG–2302

NAME

relo – Relocatable object table format under UNICOS

SYNOPSIS

```
#include <relo.h>
```

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

The assembler and the compilers generate relocatable object tables, as described in this entry. Some UNICOS commands that process relocatable object tables use internal definitions rather than the relo.h include file.

The relocatable binary tables are presented as C language code.

Compilation or assembly produces a file that contains one or more relocatable modules. To combine relocatable modules into library files, use ar(1), bld(1), or tsar(8). The ld(1) and segldr(1) loaders process the modules to create an executable file with an optional symbol table. This manual entry describes the bit fields in the relocatable binary Program Descriptor table (PDT), the Text table (TXT), the Relocation table (REL), the Extended Relocation table (XRL), and the Module Termination table (MTT). For a description of the Debug Symbol table (SMT), see the

When a set of routines is compiled or assembled and a set of relocatable object modules is produced, each object module set contains the instructions, data, and relocation information needed for linking the module, which creates an executable image.

A relocatable object file consists of a contiguous set of one or more such relocatable object modules. The relocatable modules that the compiler produces may be linked separately, or the subroutines may be merged before linking by using bld(1). Library modules are linked with the object files as needed. The loaders produce an executable file with an optional Global Symbol table (GST) to describe the global variables. Usually, the GST is joined to the end of the executable file to form one file.

Each relocatable module consists of a sequence of relocatable tables. A single relocatable module corresponds to an ident/end sequence in assembly language, a source file in C, a Fortran subroutine, or a Pascal compilation unit. The first table of each module is the PDT, which defines the blocks, entry points, and externals used in the routine. Any number of TXTs, RELs, XRLs, and SMTs may come between the beginning PDT and the ending MTT. The TXT contains the instructions and data that will be linked into the program block, and the REL and XRL contain the relocation information. The SMT describes all identifiers used in a given module. The last table of each object module is an MTT; it terminates the set of tables that define the object module for a one routine.

Each table begins with a header word (`tbl_hdr`) that contains the table type (`hdr_type`) and word count (`hdr_len`) of the table. These fields provide the record structure of the relocatable binary file and appear as the first word of every table. The table type identifies the binary table. The word-count field gives the number of words in the table. This field permits the tables to vary in length. All variable-length fields also contain, or are preceded by, a field that specifies their length. In particular, all ASCII names vary in length; each is preceded by a character count in an 8-bit field. Global names, entry points, and externals are limited to 255 characters.

In the C language representation of these tables, FIELD denotes an unsigned long variable.

Schematic Representations

A Cray Research system word contains 64 bits. You may divide each word into consecutive strings of bits, which are referred to as *bit fields*.

Table header word

The first word of each table is a table header that contains a table type code, an optional block index field, and a table length. This table header structure is used for the Program Descriptor table (PDT), Text table (TXT), Relocation table (REL), Extended Relocation table (XRL), Module Termination table (MTT), build Library (`bld`(1)), Header table (LHT), Build Library (`bld`(1)), and Directory table (BLD). It is defined as follows:

```
struct tbl_hdr {
       FIELD   hdr_type  :  7;   /* Table type                 */
       FIELD             :  9;   /* (Unused, reserved by CRI)  */
       FIELD   hdr_bi    :  16;  /* Block index (optional)     */
       FIELD   hdr_len   :  32;  /* Table length               */
```

The constants for the table type codes, which are used in the first word of all tables, are defined as follows:

```
#define PDT_TYPE  017  /* Program descriptor table     */
#define TXT_TYPE  016  /* Text table                   */
#define REL_TYPE  015  /* Relocation table             */
#define XRL_TYPE  014  /* Extended relocation table    */
#define MTT_TYPE  010  /* Module termination table     */
#define LHT_TYPE  001  /* Library (build) header table */
#define BLD_TYPE  002  /* Build directory table        */
#define SYM_TYPE  011  /* Module symbol table          */
#define CMB_TYPE  021  /* Common block symbol table    */
#define GNT_TYPE  027  /* Global symbol table          */
```

The PDT, TXT, REL, XRL, and MTT contain the text and relocation information that defines the contents of the module. These tables are described in this section. `bld`(1) creates the LHT and BLD for library files. The SMT and CMB contain information describing the symbols within the modules. The GST resides in the executable file and contains information that describes the global symbols in an entire program. The , describes the SMT, CMB, and GST.

Program Descriptor Table (PDT)

The PDT is the first table for a routine. It contains information needed to link the module to other modules (such as a list of blocks, entry points, and externals used in the routine) and maintenance information (such as the date and time of compilation or assembly, the generating product name and version, and the generating user number). The four sections of the PDT are the PDT Header, PDT Block Names, PDT Entry Points, and PDT External Names.

PDT header

The following structure defines the fields in the Program Descriptor table (PDT) header.

```
struct pdttabl {
      FIELD pdthdr;                 /* Use tbl_hdr structure here */
      FIELD pdthdsz     :     16;   /* Word size of header area   */
      FIELD pdtblsz     :     16;   /* Word size of block area    */
      FIELD pdtensz     :     16;   /* Word size of entry area    */
      FIELD pdtexsz     :     16;   /* Word size of external area */
      FIELD pdtmdy;                 /* MM/DD/YY - this compilation     */
      FIELD pdthms;                 /* HH:MM:SS - this compilation     */
      FIELD pdtcmpid;               /* Generating product name    */
      FIELD pdtcmpvr;               /* Generating product version */
      FIELD pdtosvr;                /* Host OS version      */
      FIELD pdtudt;                 /* UNICOS time stamp (date)   */
      FIELD             :     1;    /* (Unused, reserved by CRI)  */
      FIELD pdtfe       :     1;    /* Fatal error flag (1==true) */
      FIELD pdtbd       :     1;    /* Block data module (1==true)     */
      FIELD pdtmpa      :     1;    /* Module passed address flag */
      FIELD pdtdc       :     1;    /* Dual case names flag(1==true)   */
      FIELD pdtusr      :     8;    /* (Unused, reserved for user)     */
      FIELD             :     1;    /* (Unused, reserved by CRI)  */
      FIELD pdtmf       :     2;    /* Module flag for Fortran 90:     */
                                    /*    0 = independent         */
                                    /*    1 = first of a module   */
                                    /*    2 = in a module set     */
                                    /*    3 = last of a module    */
      FIELD pdtfnl      :     8;    /* Char count in file name    */
      FIELD pdtmnl      :     8;    /* Char count in module name  */
      FIELD pdtss       :     32;   /* Stack size requirement     */
      FIELD pdtuqnm;                /* Unique ID for module name  */
      FIELD             :     32;   /* (Unused, reserved by CRI)  */
      FIELD pdtmul      :     16;   /* Length of module use field */
      FIELD pdtcmtl     :     8;    /* Length of comments field   */
      FIELD pdtmtl      :     8;    /* Length of machine type field    */
                       /* Remaining fields follow    */
```

PDT block name

The block section of the PDT contains zero or more block entries, each of which has the following format:

```
struct pdtblck {
      FIELD pdtbkcb     :     1;    /* Common block flag (1==true)     */
      FIELD pdtbkl      :     3;    /* Block location */
      FIELD pdtbkc      :     3;    /* Block contents */
      FIELD pdtbkt      :     3;    /* Block type     */
      FIELD pdtbal      :     1;    /* Block align flag     */
      FIELD pdtbef      :     1;    /* Block entry flag     */
      FIELD             :     4;    /* (Unused, reserved by CRI)  */
      FIELD pdtbusr     :     8;    /* (Unused, reserved for user)     */
      FIELD pdtbknl     :     8;    /* Char count in block name   */
      FIELD pdtbkln     :     32;   /* Block length (words) */
                        /* Block name follows   */
};
```

The block name follows the `pdtbkln` field in the minimum number of words required to store `pdtbknl` characters. This name is left justified and zero filled within this field.

The constants for the block location field (`pdtbkl`) are defined as follows:

```
#define BKL_CM    0     /* Common memory        */
#define BKL_AX    4     /* CRAY Y-MP auxiliary memory*/
```

The constants for the block contents field (`pdtbkc`) are defined as follows:

```
#define BKC_UNK   0     /* Unknown  */
#define BKC_IX    1     /* Instructions only    */
#define BKC_DT    2     /* Data only      */
#define BKC_BS    3     /* Data only with no text (bss)    */
#define BKC_CN    4     /* Constants only */
#define BKC_ZD    5     /* Data only with no text     */
            /* (zero fill assumed)  */
```

The constants for the block type field (`pdtbkt`) are defined as follows:

```
#define BKT_CM    0     /* Regular common block */
#define BKT_TCM   2     /* Task common block    */
#define BKT_DBF   4     /* Dynamic block  */
```

The constants for the block align flag (`pdtbal`) are defined as follows:

```
#define BAL_NA    0     /* No alignment   */
#define BAL_AL    1     /* Align to instruction buffer boundary  */
```

PDT Entry Point

The entry point section of the PDT contains zero or more entries, each of which has the following format:

```
struct pdtent {
      long  pdtepv;                 /* Entry point (signed) value */
      FIELD pdtepf      :     1;    /* Primary entry flag (1==true)    */
      FIELD           : 1;    /* (Unused, reserved by CRI)  */
      FIELD pdtenl      :     8;    /* Char count in entry name   */
      FIELD pdterm      :     3;    /* Suggested relocation mode  */
      FIELD           : 27;   /* (Unused, reserved by CRI)  */
      FIELD pdteusr     :     8;    /* (Unused, reserved for user)     */
      FIELD pdtebi      :     16;   /* Block index    */
                              /* Entry name follows   */
};
```

The entry name follows the block index field (`pdtebi`) in the minimum number of words required to store `pdtenl` characters. This name is left justified and zero filled within this field.

The constants for the suggested relocation mode field (`pdterm`) are defined as follows:

```
   #define RM_WORD   0     /* Word address   */
   #define RM_HALF   1     /* Half word address    */
   #define RM_PARC   2     /* Parcel address */
   #define RM_BYTE   3     /* Byte address   */
   #define RM_BIT    6     /* Bit address    */
   #define RM_ENTR   7     /* Relocation mode from */
               /* associated entry (pdterm); */
               /* legal on external references only.    */
```

PDT External

The externals section of the PDT contains zero or more entries, each of which has the following format:

```
struct pdtext  {
      FIELD pdtxmn      :     1;    /* Module specification */
      FIELD           : 1;    /* (Unused, reserved by CRI)  */
      FIELD pdtxnl      :     8;    /* Char count in external name     */
      FIELD pdtxsf      :     1;    /* Soft external (1==true)    */
      FIELD pdtxct      :     2;    /* Call tree information      */
      FIELD pdtxpa      :     1;    /* External passed as argument     */
      FIELD           : 42;   /* (Unused, reserved by CRI)  */
      FIELD pdtxusr     :     8;    /* (Unused, reserved for user)     */
                              /* External name follows     */
};
```

The external name follows the `pdtxusr` field in the minimum number of words required to store `pdtxnl` characters. This name is left justified and zero filled within this field.

The constants for the call tree information field (pdtxct) are defined as follows:

```
#define XCT_EXT   0     /* Regular external     */
#define XCT_THDO  1     /* External is task head      */
```

Text Table (TXT)

The TXT follows the PDT; any number of TXTs can be after the PDT and before the MTT. You may intermix RELs, XRLs, and TXTs, but TXTs should precede RELs and XRLs that relocate the locations filled in by the TXT. The TXT contains the instructions and data to be linked into the program. The TXT begins with the table header word tbl_hdr at word 0; the hdr_type field identifies it as a text table. One or more item entries follow the header word.

```
struct txtitem {
      long  txtinc      :     17;   /* Incr between dups (signed) */
      FIELD txtsba      :     38;   /* Starting bit address */
      FIELD txtnbl      :     6;    /* Number of bits in last word      */
      FIELD txtusr1     :     3;    /* (Unused, reserved for user)      */
      FIELD txtusr2     :     5;    /* (Unused, reserved for user)      */
      FIELD           : 8;    /* (Unused, reserved by CRI)  */
      FIELD txtndp      :     19;   /* Number of duplications     */
      FIELD txtntw      :     32;   /* Number of text words */
                              /* Text words follow    */
};
```

The text words immediately follow the item header.

Relocation Table (REL)

Any number of REL tables may be between the PDT and the MTT. You may intermix RELs, XRLs, and TXTs; but RELs should follow any TXTs that fill in the locations relocated by the REL. The REL contains relocation information for the module. The REL begins with the table header word tbl_hdr at word 0; the hdr_type field identifies it as a relocation table.

```
struct relitem {
      FIELD relrt   :   1;    /* Relocation type      */
      FIELD relri   :   16;   /* Relocation index     */
      FIELD relrba  :   38;   /* Rightmost bit address      */
      FIELD relfl   :   6;    /* Field length in bits */
                              /* to relocate    */
      FIELD relrm   :   3;    /* Relocation mode      */
};
```

The constants for the relocation type field (relrt) are defined as follows:

```
#define RT_BLK    0     /* Block entry    */
#define RT_EXT    1     /* External entry */
```

The constants for the relocation mode field (relrm) are defined as follows:

```
#define RM_WORD   0     /* Word address   */
#define RM_HALF   1     /* Half word address    */
#define RM_PARC   2     /* Parcel address */
#define RM_BYTE   3     /* Byte address   */
#define RM_BIT    6     /* Bit address    */
#define RM_ENTR   7     /* Relocation mode from */
            /* associated entry (pdterm); */
            /* legal on external references only.    */
```

Extended Relocation Table (XRL)

Any number of XRLs may be between the PDT and MTT. RELs, XRLs, and TXTs, but XRLs should follow any TXTs that fill in locations relocated by the XRL. The XRL contains the relocation information for the module. The XRL begins with the table header word `tbl_hdr` at word 0; the `hdr_type` field identifies it as an extended relocation table. The XRL resembles the REL with the addition of the `xrlusr`, `xrln`, and `xrlsr` fields.

```
struct xrlitem {
      FIELD xrlrt   :   1;    /* Relocation type      */
      FIELD xrlri   :   16;   /* Relocation index     */
      FIELD xrlusr  :   8;    /* (Unused, reserved for user)     */
      FIELD         :   23;   /* (Unused, reserved by CRI)  */
      FIELD xrlsp   :   3;    /* Special relocation   */
      FIELD xrln    :   1;    /* Sign before relocation    */
      FIELD xrlsr   :   3;    /* Sign specification of result    */
      FIELD xrlfl   :   6;    /* Field length in bits */
                              /* to relocate    */
      FIELD xrlrm   :   3;    /* Relocation mode      */
      FIELD         :   26;   /* (Unused, reserved by CRI)  */
      FIELD xrlrba  :   38;   /* Rightmost bit address     */
};
```

The constants for the extended relocation type field (`xrlrt`) are defined as follows:

```
#define RT_BLK    0     /* Block entry    */
#define RT_EXT    1     /* External entry */
```

The constants for the special relocation field (`xrlsp`) are defined as follows:

```
#define SP_NONE   0     /* No special relocation      */
#define SP_RVHF   1     /* Reversed halves relocation */
#define SP_3PRL   2       /* Three parcel relocation      */
```

The constants for the sign specification of result field (`xrlsr`) are defined as follows:

```
#define SR_NONE   0     /* Sign does not matter */
#define SR_POS    1     /* Field must be positive     */
#define SR_NEG    2     /* Field must be negative     */
#define SR_EXT    3     /* Field is sign extended     */
```

The constants for the extended relocation mode field (`xrlrm`) are defined as follows:

```
#define RM_WORD   0     /* Word address   */
#define RM_HALF   1     /* Half word address    */
#define RM_PARC   2     /* Parcel address */
#define RM_BYTE   3     /* Byte address   */
#define RM_BIT    6     /* Bit address    */
#define RM_ENTR   7     /* Relocation mode from */
            /* associated entry (pdterm); */
            /* legal on external references only.     */
```

Module Termination Table (MTT)

The MTT is at the end of the relocatable binary module. The MTT terminates the set of tables defining the object module for one routine. The MTT begins with the table header word `tbl_hdr` at word 0; the `hdr_type` field identifies it as an MTT.

The MTT is defined by the following structure.

```
struct mtttabl {
      FIELD mtthdr;     /* Use tbl_hdr structure here */
      FIELD mttcksm;    /* Checksum */
};
```

FILES

`/usr/include/relo.h`	Format of relocatable object tables

SEE ALSO

`symbol`(5) for a description of the UNICOS symbol table entry format

`ar`(1) to invoke the archive and library maintainer for portable archives
`bld`(1) to maintain relocatable libraries
`cc`(1) to invoke the Cray Standard C compiler
`date`(1) to print and set the date
`ed`(1) to invoke the text editor
`ld`(1) to invoke the link editor with traditional UNIX invocation
`pascal`(1) to invoke the Pascal compiler
`segldr`(1) to invoke the Cray Research segment loader (SEGLDR)
in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

`tsar`(8) to invoke the system data processing language processor
in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

resolv.conf – Domain name resolver configuration file

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The resolver configuration file /etc/resolv.conf contains information that the resolver routines read the first time they are invoked by a process. The file is human readable and contains a list of keywords with values that provide various types of resolver information.

If the only name server to be used is the local server and the host name, configured elsewhere through the hostname(1) command, is the fully qualified domain name, the resolv.conf file probably is not needed. Otherwise, configure this file to specify the name servers, local domain, and other optional configuration information.

The configuration options are as follows:

domain local.domain

The domain keyword designates local.domain as the default domain for queries that are not fully qualified. The local.domain parameter is appended to unqualified domain names in an attempt to form a fully qualified domain name. Most queries for names within this domain can use short names, relative to the local domain. If no domain entry is present, the domain is determined from the local host name returned by gethostname(2); the domain part is everything after the first period (.). Finally, if the host name does not contain a domain part, the root domain is assumed.

nameserver address

The nameserver keyword designates a name server that answers domain name queries for this machine. The address parameter specifies the Internet address (in dot notation) of a local or remote server that the resolver should query. You can list up to MAXNS (currently 3) name servers, one per keyword. If multiple servers exist, the resolver library queries them in the order listed. If no name server entries are present, the default uses the name server on the local machine. The algorithm used is to try a name server, and if the query times out, try the next, and so on until all name servers are tried; then repeat trying all of the name servers until a maximum number of retries are made.

options option

The options keyword designates internal resolver variable settings to be modified. The option parameter may be one of the following:

debug Sets RES_DEBUG in res.options.

`ndots:`*n* Sets a threshold for the number of dots that must appear in a name given to `res_query`(3C) before an initial absolute query is made. The default for *n* is 1, meaning that if any dots are in a name, the name is tried first as an absolute name before any search list elements are appended to it.

`search search.list`

The `search` keyword designates `search.list` as a set of domains to try when attempting to resolve a domain name. By default, `search.list` contains only the local domain name. This may be changed by listing the desired domain search path following the `search` keyword with spaces or tabs separating the names. Most resolver queries are tried using each component of the search path in turn until a match is found. This process is slow and generates a lot of network traffic if the servers for the listed domains are not local. Queries will time out if no server is available for one of the domains. The `search list` is currently limited to six domains and a total of 256 characters.

`sortlist address.list`

The `sortlist` keyword designates a preferred ordering of addresses returned by `gethostbyname`(3C). The `gethostbyname`(3C) call returns addresses that match one of the `address.list` entries before those that do not match. The `address.list` specifies a set of IP address `netmask` pairs. The `netmask` is optional and defaults to the natural `netmask` of the net. The IP address and optional network pairs are separated by slashes. You may specify up to 10 pairs.

The following example returns addresses on the `130.155.160.0/255.255.240.0` network first, then addresses on the `130.155.0.0` network, and finally, other addresses:

```
sortlist 130.155.160.0/255.255.240.0 130.155.0.0
```

The `domain` and `search` keywords are mutually exclusive. If more than one instance of these keywords is present, the last instance wins.

To override the `search` keyword of a system's `resolv.conf` file on a per process basis, set the `LOCALDOMAIN` environment variable to a space-separated list of search domains.

To amend the `options` keyword of a system's `resolv.conf` file on a per process basis, set the `RES_OPTIONS` environment variable to a space-separated list of resolver options.

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

EXAMPLES

The following example shows a file that lists one local and two remote name servers and establishes a default domain name of `ourdomain.com`:

```
nameserver  127.0.0.1
nameserver  123.45.67.89
nameserver  234.56.78.90
domain      ourdomain.com
```

FILES

`/etc/resolv.conf` Domain name query reference file

SEE ALSO

`gethostname`(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

`gethostbyname`(3C) (see `gethost`(3C)), `res_query`(3C) (see `resolver`(3C)) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

`named`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

UNICOS Networking Facilities Administrator's Guide, Cray Research publication SG–2304

NAME

rhosts – Specifies a list of trusted remote hosts and account names

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The .rhosts file lets you specify a list of remote hosts and account names (users) that may log in to your account free of the normal user validation. Remote account names that are listed in your .rhosts file may do the following:

- Log in (using rlogin(1B)) to the local host with your local account name without being asked to provide the password for that account
- Copy files (using rcp(1)) between the remote host and the local host, and vice versa
- Execute commands remotely (using remsh(1B)) on the local host from a remote host

The .rhosts file is an optional file. If present, it must be in your home directory on the local host, it must be owned by either you or the super user (root), and it must not be world or group writable.

Your .rhosts file is checked only after a remote login request is not matched by an entry in /etc/hosts.equiv (see hosts.equiv(5)). Each entry in .rhosts identifies a remote host and an account name on that host. If either of these fields is not matched by an incoming request, that entry is not matched.

If an entry in .rhosts contains only the name of a remote host, a request coming from that remote host will be matched only if the remote account name is the same as your local account name. If none of the entries is matched, automatic login is denied; you are then prompted for a password (unless you used rsh(1B), in which case, rsh displays the message Permission denied and closes the connection).

An * symbol in .rhosts allows your account to perform from any remote host the functions that .rhosts controls.

The format of an entry in .rhosts is as follows:

remote_host
or
remote_host remote_account_name
or
*

You must separate *remote_host* from *remote_account_name* by one space.

NOTES

Use of the `.rhosts` file presents a security risk. In situations in which security is a concern, use the file very cautiously or not at all.

The system configuration may require the `/etc/hosts.equiv` and `.rhosts` files each to contain a match for the remote host, and it also may require the remote user and local user names to match.

MESSAGES

The following error message may occur:

`permission denied`
: The `.rhosts` file must not be owned by another user or writable by the world or group. If it is, the `.rhosts` file will not be read, and this message will appear.

SEE ALSO

`hosts.equiv`(5)

`rcp`(1), `remsh`(1B), `rlogin`(1B) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

`rcmd`(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

`rlogind`(8), `rshd`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

TCP/IP Network User's Guide, Cray Research publication SG–2009

NAME

rmtab – List of remotely mounted file systems

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The /etc/rmtab file contains a list of all file systems on this machine that have been mounted remotely by other machines. Whenever a file system is mounted remotely, the machine providing the file system makes an entry in rmtab.

The rmtab file is a series of lines that has the following format:

hostname:*directory*

This file is used only for administrator information. The system does not use it during remote mount operations.

BUGS

The rmtab table is not always completely accurate.

FILES

/etc/rmtab List of remotely mounted file systems

SEE ALSO

mount(8), mountd(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

`sccsfile` – Source Code Control System (SCCS) file format

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

An SCCS file is an ASCII file that contains control and data information to record multiple versions of a single ASCII source file. The SCCS file consists of six logical parts:

Checksum	Sum of all characters in the file except those of the first line
Delta table	Information about each delta
User names	Login names or numerical group IDs of users who may add deltas
Flags	Definitions of internal keywords
Comments	Users' descriptive information about the file
Body	Actual text lines intermixed with control lines

Throughout an SCCS file, there are lines that begin with the ASCII SOH (start of heading) character (octal 001). This character is hereafter referred to as the *control character* and is represented graphically as "@". A line of user-supplied text may not begin with the control character.

Each logical part of an SCCS file is described in detail by the following. Entries of the form *DDDDD* represent a 5-digit string (a number between 00000 and 99999).

Checksum — The checksum is the first line of an SCCS file. The form of the line is as follows:

@h*DDDDD*

The value of the checksum is the sum of all characters except those of the first line. The @h characters provide a magic number for SCCS.

Delta table — The delta table of an SCCS file consists of one or more entries, each of which contains information about one version of the source file. Each entry in the delta table has the following format:

```
@s DDDDD/DDDDD/DDDDD
@d type SCCS-ID yy/mm/dd hh:mm:ss pgmr DDDDD DDDDD
@i DDDDD ...
@x DDDDD ...
@g DDDDD ...
@m MR-number
      ...
@c comments ...
      ...
@e
```

Each of these entries is described as follows:

`@s` *DDDDD/DDDDD/DDDDD*
: Number of lines inserted, deleted, and unchanged, respectively.

`@d` *type SCCS-ID yy/mm/dd hh:mm:ss pgmr DDDDD DDDDD*
: Type of the delta (currently, `D` = normal and `R` = removed), the SCCS ID of the delta, the date and time of creation of the delta, the login name that corresponds to the real user ID at the time the delta was created, and the serial numbers of the delta and its predecessor, respectively.

`@i` *DDDDD ...*
: Serial numbers of deltas included; this line is optional.

`@x` *DDDDD ...*
: Serial numbers of deltas excluded; this line is optional.

`@g` *DDDDD ...*
: Serial numbers of deltas ignored; this line is optional.

`@m` *MR-number*
: Modification request (MR) number associated with the delta; this line is optional. More than one `@m` line can exist, each containing one MR number.

`@c` *comments ...*
: User-supplied comments associated with the delta. This line is optional; more than one `@c` line can exist.

`@e`
: End of the delta table entry.

User names
: The list of login names or numerical group IDs of users who may add deltas to the file, one name to a line. The lines that contain these login names and numerical group IDs are surrounded by the bracketing lines `@u` and `@U`. They may not begin with the control character. An empty list allows any user to make a delta. Any line that starts with a `!` prohibits the succeeding group or user from making deltas.

Flags
: Flags are keywords used internally (for more information on their use, see `admin`(1)). Each flag line takes the following form:

`@f`*flag optional text*

The flags are defined as follows:

@f`t` *type of program*
Defines the replacement for the `%Y%` identification keyword.

@f`v` *program name*
Controls prompting for MR numbers in addition to prompting for comments; if *program name* is present, it defines an MR number validity-checking program that is called when making changes to the SCCS file.

@f`i` *keyword string*
Controls the warning/error aspect of the ''No id keywords'' message. When the `i` flag is not present, this message is only a warning; when the flag is present, this message causes a fatal error (the file is not retrieved or the delta is not made).

@f`b` Causes a branch in the delta tree when used with the `-b` keyletter of the SCCS `get`(1) command.

@f`m` *module name*
Defines the first choice for the replacement text of the `%M%` identification keyword.

@f`f` *floor*
Defines the *floor* release; that is, the release below which no deltas may be added.

@f`c` *ceiling*
Defines the *ceiling* release; that is, the release above which no deltas may be added.

@f`d` *default SID*
Defines the default SCCS ID to be used when none is specified on a `get` command.

@f`n` Causes the command `delta`(1) to insert a *null* delta (a delta that applies no changes) in releases skipped when a delta is made in a new release (for example, when delta 5.1 is made after delta 2.7, releases 3 and 4 are skipped). The absence of the `n` flag causes skipped releases to be completely empty.

@f`j` Causes the SCCS `get` command to allow concurrent edits of the same base SCCS ID.

@f`l` *lock releases*
Defines a list of releases that are locked against editing (the SCCS `get` command with the `-e` keyletter).

@f`q` *user-defined text*
Defines the replacement for the `%Q%` identification keyword.

@fz *application name*
Reserved for use in certain specialized interface programs.

Comments — User-supplied comments are surrounded by the bracketing lines @t and @T. This comment information is sometimes called *descriptive text*. It is separate from the per-delta comments in the delta table and is sometimes used to describe the purpose of the source file. These comment lines cannot begin with the control character.

Body — The body consists of text lines and control lines. Text lines may not begin with the control character. There are three kinds of control lines: insert, delete, and end. *DDDDD* is the serial number that corresponds to the delta for the control line.

@I *DDDDD*
Insert

@D *DDDDD*
Delete

@E *DDDDD*
End

SEE ALSO

admin(1), delta(1), get(1), prs(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

NAME

sectab – Format for table of defined security names and values

SYNOPSIS

```
#include <sys/sectab.h>
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The sectab structure is used to hold security names and associated values. This structure is defined as follows:

```
#define MAXNAMES 64
#define MAXNAMELEN 256

struct sectab {
        char tb_name[MAXNAMES][MAXNAMELEN];     /* Security names */
        long tb_num[MAXNAMES + 1];              /* Security name values /*
};
```

The getsectab(2) system call uses the sectab structure to hold a maximum of 64 security name strings, each of which may consist of 255 characters plus a NULL terminator. It also holds a maximum of 64 values that are associated with the security names. getsectab(2) terminates the list of values with –1.

FILES

/usr/include/sys/sectab.h

SEE ALSO

getsectab(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

NAME

sem – Semaphore facility

SYNOPSIS

```
#include <sys/sem.h>
```

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4

DESCRIPTION

The sem man page describes the constants and structures in the sys/sem.h include file.

The following semaphore operation flag can be specified:

SEM_UNDO Sets up adjust on exit entry.

The command definitions for the semctl(2) system call are as follows:

GETNCNT Gets semncnt.

GETPID Gets sempid.

GETVAL Gets semval.

GETALL Gets all cases of semval.

GETZCNT Gets semzcnt.

SETVAL Sets semval.

SETALL Sets all cases of semval.

The semid_ds structure contains the following members:

```
struct ipc_perm      sem_perm     /* operation permission structure */
unsigned short int   sem_nsems    /* number of semaphores in set */
time_t               sem_otime    /* last semop(2) time */
time_t               sem_ctime    /* last time changed by semctl(2) */
```

The pid_t, time_t, key_t, and size_t types are defined as described in sys/types.h.

A semaphore is represented by a structure that contains the following members:

```
unsigned short int   semval     /* semaphore value */
pid_t                sempid     /* process ID of last operation */
unsigned short int   semncnt    /* number of processes waiting for semval
                                   to become greater than current value */
unsigned short int   semzcnt    /* number of processes waiting for semval
                                   to become zero */
```

The structure `sembuf` contains the following members:

```
unsigned short int   sem_num    /* semaphore number */
short int            sem_op     /* semaphore operation */
short int            sem_flg    /* operation flags */
```

The following are declared as functions and also may be defined as macros:

```
int  semctl (int semid, int semnum, int cmd ...);
int  semget (key_t key, int nsems, int semflg);
int  semop (int semid, struct sembuf *sops, size_t nsops);
```

When this header file is included, all of the symbols from `sys/ipc.h` also will be defined.

SEE ALSO

`ipc`(5), `types`(5)

`semctl`(2), `semget`(2), `semop`(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

`ipc`(7) Online only

NAME

sendmail.cf – Configuration file for TCP/IP mail service

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The sendmail(8) program works with three mail services: mail(1), mailx(1), and the DARPA Simple Mail Transfer Protocol (SMTP) (see sendmail(8)).

The usr/lib/sendmail.cf file is a cryptic set of rules and definitions that the sendmail(8) program uses to determine the next step a mail message should take toward its destination and to transfer the mail message to the next step.

Although the actual determination is at the discretion of the system administrator or the author of the contents of the sendmail.cf file, mail messages with recipients specified by a DARPA-style address (for example, user@host) traditionally are delivered to the destination host by using SMTP, and mail messages addressed to local users are delivered through a local mail delivery agent. (Under UNICOS, this delivery agent is the mail(1) program.)

For information on configuring the sendmail.cf file, see *UNICOS Networking Facilities Administrator's Guide*, Cray Research publication SG–2304.

FILES

/usr/lib/sendmail.cf	TCP/IP mail handler file

SEE ALSO

mail(1), mailx(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

sendmail(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

UNICOS Networking Facilities Administrator's Guide, Cray Research publication SG–2304

DARPA Internet Request for Comments, RFC 819, RFC 821, and RFC 822

NAME

services – Network service name database

SYNOPSIS

/etc/services

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The /etc/services file contains the database of known services available in the network.

TCP/IP entries:

For each service, one line in the services file should contain the official service name, the port number, the protocol name, and any aliases that exist for the service name. Items are separated by any number of blanks, tab characters, or a combination of both. The port number and protocol name are considered one item; use a / symbol to separate the port and protocol specified (for example, 512/tcp).

A # symbol indicates the beginning of a comment; if you specify this symbol, routines that search the file do not interpret additional characters up to the end of the line.

Service names may contain any printable character other than a field delimiter, newline, or comment (#).

When you modify TCP/IP entries that have privileged port numbers (512 to 1023), use the rsvportbm(8) command to update the kernel's reserved port table. The bindresvport(3C) and rresvport(3C) routines query the kernel table to ensure that a port in the /etc/services file is not used. The rsvportbm(8) command usually is run at system startup.

EXAMPLES

The following example shows sample entries for /etc/services:

```
#
# Network services, Internet style
#
ftp        21/tcp
telnet     23/tcp
smtp       25/tcp   mail
hostnames  101/tcp  hostname  # usually from sri-nic
sunrpc     111/udp
sunrpc     111/tcp
#
# Host specific functions
#
tftp       69/udp
finger     79/tcp
#
# UNIX specific services
#
exec       512/tcp
login      513/tcp
shell      514/tcp  cmd  # no passwords used
talk       517/udp
```

FILES

`/etc/services` Network database file

SEE ALSO

`getserv`(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

`rsvportbm`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

share – Fair-share scheduler parameter table

SYNOPSIS

```
#include <sys/share.h>
```

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

The kernel uses the share structure to define and retain the variables used as constants and feedback values in the calculations performed by the fair-share scheduler.

The system administrator can manipulate the values in the share structure that are treated as constants by using the shradmin(8) administrator command.

The format of the share structure is defined in the sys/share.h include file, as follows:

```
/*
**     Share scheduling parameters
*/

struct sh_consts
{
     /** Parameters **/

     int     sc_fl;              /* Scheduling flags */
     int     sc_delta;           /* Run rate for scheduler in secs. */
     int     sc_mxusers;         /* Max. number of active users */
     int     sc_mxgroups;        /* Max. group nesting */
     float   sc_ratedecay;       /* Decay rate for ‘‘kl_rate’’ */
     float   sc_mxpri;           /* Max. absolute priority */
     float   sc_mxupri;          /* Max. priority for a normal process */
     float   sc_mxusage;         /* Max. usage considered */
     float   sc_decay;           /* Decay factor for ‘‘kl.l_usage’’ */
     int     sc_syscall;         /* Cost of system call */
     int     sc_bio;             /*   "  "  logical block i/o */
     int     sc_tio;             /*   "  "  stream i/o */
     int     sc_tick;            /*   "  "  cpu tick */
     int     sc_click;           /*   "  "  memory tick */
     float   sc_basepridecay;    /* Base for decay for sharepri */
     float   sc_pridecay;        /* Decay rate for maximally niced processes */
     float   sc_maxushare;       /* Factor for max effective user share */
     float   sc_mingshare;       /* Factor for min effective group share */
```

```
float   sc_percent;          /* Current charging %.  1.0 = 100% */
float   sc_sharemin;         /* Minimum user share */
float   sc_pspare[2];        /* <spare> */
uint    sc_syncsec:32,       /* Sync lnodes with UDB every N secs */
        sc_procmax:32;       /* Maximum number of processes  */
int     sc_memmax;           /* Maximum aggregate mem_clicks */

          /** Feedback **/

          int     sc_users;            /* Number of active users */
          int     sc_groups;           /* Number of active groups */
          float   sc_highshpri;        /* High value of p_sharepri */
          float   sc_mxcusage;         /* Max. current usage */
          int     sc_csyscall;         /* Count system calls */
          int     sc_cbio;             /*   "   logical block i/os */
          int     sc_ctio;             /*   "   stream i/os */
          int     sc_ctick;            /*   "   cpu ticks */
          int     sc_cclick;           /*   "   memory ticks */
          float   sc_fspare[2];        /* <spare> */
    };

    #ifdef     KERNEL
    extern struct sh_consts     shconsts;
    #endif

    #define     DecayRate     shconsts.sc_ratedecay
    #define     DecayUsage    shconsts.sc_decay
    #define     LASTPARAM     shconsts.sc_pspare[0]
    #define     MAXGROUPS     shconsts.sc_mxgroups
    #define     MAXSHAREPRI   shconsts.sc_mxpri
    #define     MAXUPRI       shconsts.sc_mxupri
    #define     MAXUSAGE      shconsts.sc_mxusage
    #define     MAXUSERS      shconsts.sc_mxusers
    #define     MAXUSHARE     shconsts.sc_maxushare
    #define     MaxSharePri   shconsts.sc_highshpri
    #define     MaxUsage      shconsts.sc_mxcusage
    #define     MINGSHARE     shconsts.sc_mingshare
    #define     SHARE_MIN     shconsts.sc_sharemin
    #define     PriDecay      shconsts.sc_pridecay
    #define     PriDecayBase  shconsts.sc_basepridecay
    #define     Shareflags    shconsts.sc_fl
```

```
/*
**     Share scheduling flags
*/
#define     NOSHARE          01   /* Don't run scheduler at all */
#define     ADJGROUPS        02   /* Adjust group usages */
#define     LIMSHARE         04   /* Limit maximum share */
#define     SHAREBYACCT      010  /* Share base on acct# */
#define     NOSCHED          020  /* Don't use FSS to schedule CPUs */
#define     ALLOWDEFSHARE    040  /* Allow default shares from setshare() */
#define     USRLEVLFSS      0100  /* Let shrdaemon calculate lnode values */

      /*
      **     Weighting factors for calculation of process rates
      */
      #define     RUN_WT           1.0  /* factor for runnable process */
      #define     SSLP_WT          0.6  /* factor for soft sleepers */
      #define     SWP_WT           0.2  /* factor for swapped process */

      #ifdef     KERNEL
      /*
      **     Table for pre-calculated priority decays
      */

      extern float          NiceDecays[];

      /*
      **     Table for pre-calculated rate increments
      */

      extern float          NiceRates[];

      /*
      **     Table for pre-calculated tick costs
      */

      extern int          NiceTicks[];
      #endif
```

The `share.h` structure is used in the `/usr/src/uts/md/lowmem.c` file to define the share constants and feedback-variables table.

You can retrieve the shconsts structure by using the policy(2) system call by using policy(FAIR_SHARE,GET_COSTS), and the privileged user can set it by using policy(FAIR_SHARE,SET_COSTS). The privileged user can set the SC_MXUSAGE field by using policy(FAIR_SHARE,MOD_MXUSG).

FILES

/usr/include/sys/share.h	Kernel user limits structure

SEE ALSO

lnode(5)

shrview(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

limits(2), policy(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

shradmin(8), shrdaemon(8), shrlimit(8), shrmon(8), shrtree(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

UNICOS Resource Administration, Cray Research publication SG–2302

NAME

shells – List of available user shells

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The /etc/shells file contains a list of shells (command interpreters) that are available under UNICOS. These shell names are used with the chsh(1B) and ftp(1B) commands.

The file is formatted with the full path name of each shell on a separate line.

Any line that does not begin with the full path name of a shell (that is, that does not have a slash character in the first column) is ignored. (Traditionally, however, the # symbol is used in the first column to indicate a line of comment.) Also, a white-space character (space or tab) or the comment symbol # following a full path name indicates that the remainder of the line is ignored as a comment.

EXAMPLES

An example of a /etc/shells file follows:

```
#  List of acceptable shells for chsh and ftp;
#  ftpd will not allow users to connect who do not have one of these shells
#
#  The POSIX shell
/bin/sh
#  The C shell
/bin/csh
#  The Korn shell
/bin/ksh
```

FILES

/etc/shells	File that contains a list of available shells.

SEE ALSO

chsh(1B), csh(1), ftp(1B), sh(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

NAME

shm – Shared memory facility

SYNOPSIS

```
#include <sys/shm.h>
```

IMPLEMENTATION

CRAY T90 series

STANDARDS

POSIX, XPG4

DESCRIPTION

The shm man page describes the symbolic constants and the shmid_ds structure in the sys/shm.h include file.

SHM_RDONLY Attaches read-only (defaut is read-write).

SHM_RND Rounds attach address to SHMLBA.

SHMLBA Specifies segments low boundary address multiple.

The following data types are defined through typedef:

shmatt_t Unsigned integer used for the number of times the segment is currently attached. It must be able to store values at least as large as a type unsigned short.

The shmid_ds structure contains the following members:

```
struct ipc_perm     shm_perm    /* operation permission structure */
int                 shm_segsz   /* size of segment in bytes */
pid_t               shm_lpid    /* process ID of last shared memory operation */
pid_t               shm_cpid    /* process ID of creator */
shmatt_t            shm_nattch  /* number of current attaches */
time_t              shm_atime   /* time of last shmat(2) */
time_t              shm_dtime   /* time of last shmdt(2) */
time_t              shm_ctime   /* time of last change by shmctl(2) */
```

The pid_t, time_t, key_t, and size_t types are defined as described in sys/types.h.

The following are declared as functions and also may be defined as macros:

```
void  *shmat (int shmid, const void *shmaddr, int shmflg);
int   shmctl (int shmid, int cmd, struct shmid_ds *buf);
int   shmdt  (const void *shmaddr);
int   shmget (key_t key, size_t size, int shmflg);
```

When this header file is included, all of the symbols from `sys/ipc.h` also will be defined.

SEE ALSO

`ipc`(5), `types`(5)

`shmat`(2), `shmctl`(2), `shmdt`(2), `shmget`(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

`ipc`(7) Online only

NAME

slrec – Security log record format

SYNOPSIS

```
#include <sys/types.h>
#include <sys/utsname.h>
#include <sys/secparm.h>
#include <sys/slrec.h>
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

System security information is recorded in a security log. The *security audit trail* is a set of records that documents processing and aids in tracing individual user transactions.

Every security log record has a header. There are two types of security log headers: the header used on pre-UNICOS 8.0 systems and the expanded header introduced in UNICOS 8.0, which includes subject and object compartments. The pre-UNICOS 8.0 version was retained for compatibility reasons; it also is used for all slgentry(2) calls issued by commands that have not been modified to use the new header format. The pre-UNICOS 8.0 header is defined as follows:

```
struct slghdr0 {
       time_t sl_time;        /* time (seconds since '70) */
       int    sl_uid   : 24;  /* subjects uid */
       int    sl_gid   : 24;  /* subjects gid */
       int    sl_len   : 16;  /* record length in bytes inc header */
       int    sl_ruid  : 24;  /* subjects real uid */
       int    sl_rgid  : 24;  /* subjects real gid */
       int    sl_slvl  :  8;  /* subject's level */
       int    sl_olvl  :  8;  /* object's level */
       int    sl_type  :  8;  /* record type */
       int    sl_scls  :  8;  /* subject's integrity class (obsolete) */
       int    sl_jid   : 24;  /* subject's unique jid */
       int    sl_pid   : 24;  /* subject's unique pid */
       int    sl_slog  : 32;  /* magic identifier */
       int    sl_subt  :  4;  /* record_subtype */
       int    sl_version: 4;  /* Version ID */
       int    sl_juid  : 24;  /* job owner uid */
};
```

The expanded header is defined as follows:

```
struct slghdr {
              time_t  sl_time;        /* time (seconds since '70) */
              int     sl_uid  : 24;   /* subjects uid */
              int     sl_gid  : 24;   /* subjects gid */
              int     sl_len  : 16;   /* record length in bytes inc header */
              int     sl_ruid : 24;   /* subjects real uid */
              int     sl_rgid : 24;   /* subjects real gid */
              int     sl_slvl :  8;   /* subject's level */
              int     sl_olvl :  8;   /* object's level */
              int     sl_type :  8;   /* record type */
              int     sl_scls :  8;   /* subject's integrity class (obsolete) */
              int     sl_jid  : 24;   /* subject's unique jid */
              int     sl_pid  : 24;   /* subject's unique pid */
              int     sl_slog : 32;   /* magic identifier */
              int     sl_subt :  4;   /* record_subtype */
              int     sl_version:4;   /* Version ID */
              int     sl_juid : 24;   /* job owner uid    */
              long    sl_scomp;       /* subject compartments */
              long    sl_ocomp;       /* object compartments */
};

#define SLG_MAGIC       016333067547    /* slog magic identifier */
```

The *subject* is a validated user. The *object* is a file, directory, block, character special file, FIFO special file (named pipe), socket, message, or process.

The following list summarizes the security log record types (you can find the format of each type in the `sys/slrec.h` file):

Record type	Description
`SLG_GO`	Security log start record.
`SLG_STOP`	System stop record.
`SLG_TCHG`	Time change record.
`SLG_CCHG`	System configuration change record.
`SLG_DISC`	Discretionary access record. Used on pre-7.0 UNICOS MLS systems.
`SLG_DISC_7`	Discretionary access record. Used on 7.0 and later security systems. Record includes requested access mode, which is not included in the `SLG_DISC` record.
`SLG_MAND`	Mandatory access record. Used on pre-7.0 UNICOS MLS systems.
`SLG_MAND_7`	Mandatory access record. Used on 7.0 and later security systems. Record includes requested access mode, which is not included in the `SLG_MAND` record.

SLG_OPER	(Deferred) Operational access record.
SLG_LOGN	Login validation process record.
SLG_NETW	Network access record.
SLG_DISKIO	(Deferred) Disk I/O record.
SLG_SSDIO	(Deferred) SSD I/O record.
SLG_TAPE	Tape I/O record.
SLG_EOJ	End-of-job record. This record documents an end-of-job event.
SLG_CHDIR	Change directory record. If you select optional path name tracking, this record is logged each time a change directory system call is executed.
SLG_SECSYS	Non-inode security system calls record (for example, `setucat`(2)).
SLG_NAMI	NAMI functions record (for example, `mkdir`(8) and `ln`(1)).
SLG_DAC	Discretionary access control change.
SLG_SETUID	`setuid`(2) system call record.
SLG_SU	`su`(1) attempts record.
SLG_IPNET	IP layer security violations record.
SLG_NFS	Cray NFS requests record.
SLG_FXFR	File transfer logging record.
SLG_NETCF	Network configuration changes record.
SLG_AUDIT	Security auditing option changes record.
SLG_NQS	NQS activity record.
SLG_NQSCF	NQS configuration changes record.
SLG_TRUST	Trusted process activity record.
SLG_PRIV	Privilege use record.
SLG_CRL	Cray/REELlibrarian activity.
SLG_OTHR	(Deferred) Special Cases...Other.

The kernel generates most entry types. However, some records are written by trusted user-level commands (for example, `login`(1)). To write these records, the `slgentry`(2) system call is used, and it accepts only the following record types, as defined by the `all_slgentry` structure:

```
union   all_slgentry {
        struct  slgcrl          crl;
        struct  slgfilexfr      filexfr;
        struct  slglogin        login;
        struct  slgnqs          nqs;
        struct  slgnqscf        nqschg;
        struct  slgsetuid       setuid;
        struct  slgtape         tape;
        struct  slgtape1        tape1;
        struct  slgudb          udbchg;
        struct  slgnal          nalchg;
        struct  slgwal          walchg;
        struct  slginterface    intfchg;
        struct  slgmap          mapchg;
        struct  slgipnet        ipnetchg;
#ifdef  PATHSIZE
        struct  slgtrust        trust;
        struct  slgtrust2       trust2;

};
```

FILES

`/usr/adm/sl/slogfile`	Security log
`/usr/include/sys/slog.h`	Security log header file
`/usr/include/sys/slrec.h`	Format of security log record
`/usr/include/sys/types.h`	Data type definition file
`/usr/include/sys/utsname.h`	System names
`/usr/src/uts//cf.SN/config.h`	UNICOS tunable constants definitions

SEE ALSO

`slog`(4)

`spset`(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

`slgentry`(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

`slogdemon`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

General UNICOS System Administration, Cray Research publication SG–2301

NAME

symbol – UNICOS symbol table entry format

SYNOPSIS

```
#include <symbol.h>
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

A symbol table is part of the relocatable binary table produced or used by UNICOS compilers, assemblers, and loaders. Symbol table format is defined in the symbol.h include file, and the , describes it in detail. Relocatable binary table format is described in the relo(5) entry. Some commands that process symbol tables use internal definitions, rather than the symbol.h include file.

UNICOS compilers produce symbol tables with module scope. These are either module symbol tables or common block symbol tables. A module symbol table is headed by a Module Table Header (structure mth, defined in symbol.h) and has the table type SYM_TYPE (see relo(5)). A common block symbol table is headed by a Common Block Table Header (structure cbt, defined in symbol.h) and has the table type CMB_TYPE (see relo(5)).

UNICOS loader produce symbol tables with global scope. These begin with an instance of the Global Symbol table (GST) Header (structure gnt, defined in symbol.h) and have the table type GNT_TYPE (see relo(5)).

The nlist(3C) library routine and the adb(1) debugger use the GST to look up global symbols.

FILES

/usr/include/symbol.h	UNICOS symbol table entry format

SEE ALSO

a.out(5), relo(5)

adb(1), cc(1), ld(1), segldr(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

nlist(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

NAME

tapereq – Tape daemon interface definition file

SYNOPSIS

```
#include <tapereq.h>
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The tapereq interface provides a mechanism that lets users send requests to the tape daemon, tpdaemon(8), through a FIFO special file (named pipe).

Each user request to the tape daemon has a different format, but they all include the reqhdr structure. The tapereq.h include file defines these requests and the reqhdr structure, as follows:

```
struct reqhdr {
       int  size;          /* size of this request  */
       int  code;          /* request code          */
       int  jid;           /* requester's job ID    */
       char qsub[16];      /* NQS batch job ID      */
       int  echo;          /* echo field            */
       char rpn[MAXPATH];  /* reply pipe name       */
};
```

All requests to the tape daemon include the name of a reply pipe through which the tape daemon sends a reply to the user's request. The user must create this pipe before issuing the request. All of the replies from the tape daemon to the user contain the rephdr structure. The replies and the rephdr structure are defined in tapereq.h, as follows:

```
struct rephdr {
       int  size;  /* size of reply  */
       int  echo;  /* echo field     */
       int  rc;    /* return code    */
};
```

A tape daemon request code, TR_INFO, is available to the user. It returns tape-specific data about the user's tape file. This request code is defined in tapereq.h; the status information provided also is defined in tapereq.h.

The error codes that the tape daemon returns are defined in the /usr/include/taperr.h file.

FILES

`/usr/include/tapedef.h`	Definitions for trace file size
`/usr/include/tapereq.h`	Tape daemon interface definition file
`/usr/include/taperr.h`	Tape daemon error codes
`/usr/spool/tape/trace.bmx`*xxx*	Tape daemon trace files

SEE ALSO

`rls`(1), `rsv`(1), `tpmnt`(1), `tprst`(1), `tpstat`(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

`tpdaemon`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

Tape Subsystem Administration, Cray Research publication SG–2307

NAME

tapetrace – Tape daemon trace file format

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The tape daemon, tpdaemon(8), produces trace files to debug and trace user processes. The trace files are ASCII files; use any UNICOS editor to edit them.

A trace file (trace.daemon) exists for the tape daemon. A trace file also exists for tracing user activity at the device level and has the form trace.bmx*xxx*.

A trace file (trace.avr_0) also exists for the automatic volume recognition process (avrproc). Trace files named trace.DUMMY*nn* also exist; *nn* is a number. These trace files log records of tables in the tape daemon before a device is assigned to a process.

The first 15 characters in a trace file contain the offset at which the tape daemon will start writing into the trace file. The rest of the trace file consists of trace records. The format of a trace record is as follows:

1. Time the record is produced (in *hh* : *mm* : *ss* format)
2. Time (in seconds) of the record produced since the system was initialized
3. Process ID of the process producing the record, which is the process ID of the tape daemon or the process ID of a child of the tape daemon
4. Name of the program producing the record
5. Name of the function producing the record
6. Trace information from the function

The tape_daemon_trace_file_size_bytes in the /etc/config/text_tapeconfig file defines the size of the trace files. When the size of a trace file has reached this value, the tape daemon wraps around to the beginning of the trace file and writes over it again.

FILES

/etc/config/text_tapeconfig	Tape subsystem configuration file
/usr/include/tapedef.h	Definitions for trace file size
/usr/spool/tape/trace.daemon	Trace file for tape daemon
/usr/spool/tape/trace.bmx*xxx*	Trace files for tape devices

SEE ALSO

`tpdaemon`(8)

Tape Subsystem Administration, Cray Research publication SG–2307

NAME

`tar` – Tape archive file format

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `tar` (tape archive) command dumps several files into one archived file.

A *tar tape* or file is a series of blocks. Each block is of size `TBLOCK` bytes. A file on the *tar tape* or file is represented by a header block that describes the file, followed by zero or more blocks that give the contents of the file. At the end of the tape, as an `EOF` indicator, two blocks are filled with binary 0's.

The blocks are grouped for physical I/O operations. Each group of *n* blocks is written by using one system call. To set *n*, use the `-b` option on the `tar`(1) command line. The default for *n* is 20 blocks for tapes and 128 blocks for disk files or pipes.

The `-b` option on the `tpmnt`(1) command determines the size of a tape record. The last group is always written at the full size; therefore, blocks after the two 0 blocks contain random data. On reading, the specified or default group size is used for the first read, but if that read returns less than a full tape block, the reduced block size is used for further reads.

The default header block is as follows:

```
#define TBLOCK    512
#define NAMSIZ    100
union hblock {
        char dummy[TBLOCK];
        struct header {
                char name[NAMSIZ];
                char mode[8];
                char uid[8];
                char gid[8];
                char size[12];
                char mtime[12];
                char chksum[8];
                char linkflag;
                char linkname[NAMSIZ];
                char magic[6];
                char version[2];
                char uname[32];
                char gname[32];
                char devmajor[8];
                char devminor[8];
                char prefix[155];
        } dbuf;
};
```

The `name` field is a null-terminated string. The `mode`, `uid`, `gid`, `size`, `mtime`, and `chksum` fields are zero-filled octal numbers in ASCII. Each field (of width *w*) contains *w*-2 digits, an ASCII space, and a null character, except `size` and `mtime`, which do not contain the trailing null. `name` is the name of the file, as specified on the `tar` command line. Files dumped because they were in a directory that was specified in the command line have the directory name as the prefix and */filename* as the suffix. `mode` is the file mode with the top bit masked off. `uid` and `gid` are the user and group numbers that own the file. `size` (in bytes) is the size of the file. Links and symbolic links are dumped with this field specified as 0. `mtime` is the modification time of the file at the time it was dumped. `chksum` is a decimal ASCII value that represents the sum of all of the bytes in the header block. When calculating the checksum, the `chksum` field is treated as if it were all blanks. `linkflag` is ASCII 0 if the file is a regular or a special file, ASCII 1 if it is an hard link, and ASCII 2 if it is a symbolic link. The name linked to, if any, is in `linkname`, with a trailing null character. `tar` may fill in the `magic`, `version`, `uname`, `gname`, `devmajor`, `devminor`, and `prefix` fields when creating an archive; otherwise, `tar` ignores these fields. They are defined solely for compatibility with the `pax ustar` format. Unused fields of the header are binary 0's (and are included in the checksum).

If you invoke `tar` by using the `-s` option, the following secure header block appears before each default header block:

```
struct sheader {
       short   h_smagic;
       short   h_slevel;
       long    h_compart;
       long    h_acldsk;
       short   h_aclcount;
       long    h_hdrvsn;
       char    h_dummy[1];
};
```

Each instance of `h_smagic` contains the constant 060606 (octal). The `h_slevel` and `h_compart` fields contain the file's security level and compartments, respectively. The `h_acldsk` field is a flag that indicates whether an access control list (ACL) has been archived for this file, and `h_aclcount` holds the number of entries in that ACL.

If you invoke `tar` with the `-sa` options, the following secure header appears immediately after the `sheader` header block:

```
struct nheader {
       short   h_nmagic;
       short   h_intcls;
       long    h_intcat;
       long    h_secflg;
       short   h_minlvl;
       short   h_maxlvl;
       long    h_valcmp;
       long    h_reserved[16];
       char    h_dummy[1];
};
```

Each instance of `h_nmagic` contains the constant 050505 (octal). The `h_intcls`, `h_intcat`, and `h_secflg` fields contain the file's integrity class (obsolete), integrity categories (obsolete), and security flags, respectively. The `h_minlvl`, `h_maxlvl`, and `h_valcmp` fields contain the device's minimum security level, maximum security level, and authorized compartments, respectively.

The first time a given inode number is dumped, it is dumped as a regular file. The second and subsequent times, it is dumped as a link instead. On retrieval, if a link entry is retrieved, but not the file to which it was linked, an error message is printed and you must manually rescan the tape to retrieve the linked-to file.

The encoding of the header is designed to be portable across machines.

BUGS

Names or link names longer than NAMSIZ produce error reports and cannot be dumped.

SEE ALSO

tar(1) to archive tape files
tpmnt(1) to request a tape mount for a tape file
in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

secstat(2) to get file security attributes
stat(2) to get file status
in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

NAME

taskcom – Task common table format

IMPLEMENTATION

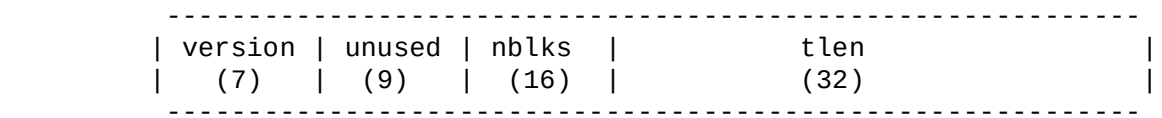
All Cray Research systems

DESCRIPTION

Task common blocks are dynamically allocated when tasks are initiated, based on information in the relocatable binary tables. The loader builds a common block, $TASKCOM, which is located in common or main memory, as a directory to the blocks in task common.

The $TASKCOM block has a one-word header. It is followed by block entries, which are followed by task common name entries. A $TASKCOM block, created by the loader, always contains at least a header entry.

The following is the format of the $TASKCOM header word:

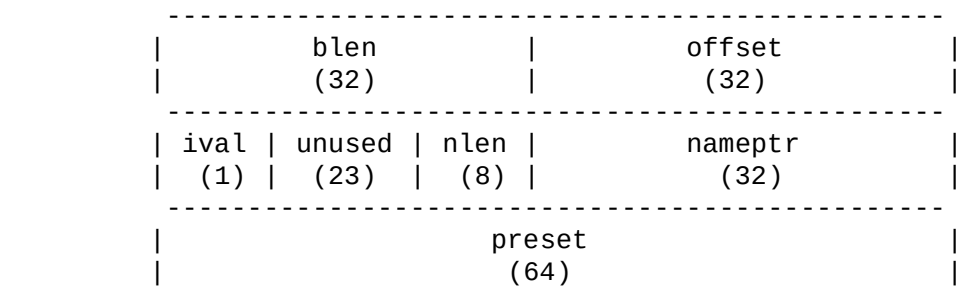


version $TASKCOM version ID (the value is 1)

nblks Number of task common blocks

tlen Total length of all task common blocks

The following is the format of a $TASKCOM block entry:



blen Number of words in this block.

offset For CRAY Y-MP systems, this is the common memory address associated with this task common block. This offset is initialized at run time to contain the actual address of this task common block's location within common memory. The loaders relocate all task common block references to the first word of the corresponding block entry within $TASKCOM.

`ival` Block initialization flag:

0 No initialization

1 Initialization

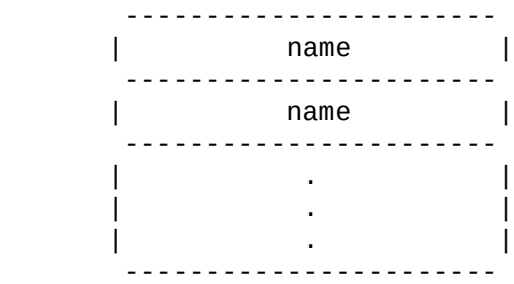
This flag is currently unused.

`nlen` Number of characters in the task common block name.

`nameptr` Word index within `$TASKCOM` of the name entry for this block. This index is relative to the base of `$TASKCOM`; that base begins with word 0, which contains the header word.

`preset` Initialization value if `ival` is set (currently unused).

The following is the format of the `$TASKCOM` name entries:



`name` ASCII name of the block, left justified and zero filled if necessary. The number of words used to contain a task common block name is (`nlen`+7)/8.

SEE ALSO

`ld`(1) to invoke the link editor with traditional UNIX invocation
`segldr`(1) to invoke the Cray Research segment loader (SEGLDR)
in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

NAME

`term` – Format of compiled `term` file

SYNOPSIS

`/usr/lib/terminfo/?/*`

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

Compiled `terminfo`(5) descriptions are placed under the `/usr/lib/terminfo` directory. To avoid a linear search of a huge UNIX system directory, a two-level scheme is used: `/usr/lib/terminfo/`*c*/*name*; *name* is the name of the terminal, and *c* is the first character of *name*. Thus, you can find `sun3` in the `/usr/lib/terminfo/s/sun3` file. Synonyms for the same terminal are implemented by multiple links to the same compiled file.

Short integers are stored in eight 8-bit bytes. The –1 value is represented by the following:

```
0377 0377 0377 0377 0377 0377 0377 0377
```

The –2 value is represented by the following:

```
0377 0377 0377 0377 0377 0377 0377 0376
```

Other negative values are illegal. The `-1` generally means that a capability is missing from this terminal. The `-2` means that the capability has been canceled in the `terminfo`(5) source and also is to be considered missing.

The compiled file is created from the source file descriptions of the terminals (see the `-I` option of `infocmp`(8)) by using the `terminfo`(5) compiler, `tic`(8), and read by the `setupterm`( ) routine. (See `curses`(3).) The file is divided into six parts: the header, terminal names, Boolean flags, numbers, strings, and string table.

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

The terminal names section comes next. It contains the first line of the `terminfo`(5) description, listing the various names for the terminal, separated by the | symbol (see `term`(7)). The section is terminated with an ASCII NUL character.

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

Between the Boolean section and the number section, 1 to 7 null bytes are inserted, if necessary, to ensure that the number section begins on an even short word boundary. All short integers are aligned on a short word boundary.

The numbers section is similar to the Boolean flags section. Each capability is stored as a short integer. If the value represented is –1 or –2, the capability is missing.

The strings section is also similar. Each capability is stored as a short integer, in the previous format. A value of –1 or –2 means the capability is missing; otherwise, the value is taken as an offset from the beginning of the string table. Special characters in `^X` or `\c` notation are stored in their interpreted form, not the printing representation. Padding information (`$<nn>`) and parameter information (`%x`) are stored intact in uninterpreted form.

The final section is the string table. It contains all of the values of string capabilities referenced in the string section. Each string is null terminated.

It is possible for `setupterm`( ) to expect a different set of capabilities than are actually present in the file. Either the data base may have been updated since `setupterm`( ) has been recompiled (resulting in extra unrecognized entries in the file) or the program may have been recompiled more recently than the database was updated (resulting in missing entries). The `setupterm`( ) routine must be prepared for both possibilities; this is why the numbers and sizes are included. You also must always add new capabilities at the end of the lists of Boolean, number, and string capabilities.

NOTES

Compiled term files from other computer systems do not have the same format as the compiled term files on Cray Research systems. Before UNICOS 7.0, terminal entries created on systems by using `tic` have a different format.

Total compiled entries cannot exceed 4096 bytes; all entries in the *name* field cannot exceed 128 bytes.

EXAMPLES

The following are examples of compiled `term` files:

```
$ infocmp sun3
sun|sun2|sun3|sun microsystems inc workstation,
        am, km, mir, msgr,
        cols#80, lines#34,
        bel=^G, clear=\f, cr=\r, cub1=\b, cud1=\n, cuf1=\E[C,
        cup=\E[%i%p1%d;%p2%dH, cuu1=\E[A, dch1=\E[P, dl1=\E[M,
        ed=\E[J, el=\E[K, ht=\t, ich1=\E[@, il1=\E[L, ind=\n,
        kcub1=\E[D, kcud1=\E[B, kcuf1=\E[C, kcuu1=\E[A,
        kf1=\EOP, kf2=\EOQ, kf3=\EOR, kf4=\EOS, khome=\E[H,
        rmso=\E[m, rs2=\E[s, smso=\E[7m,
$ od -c /usr/lib/terminfo/s/sun3 +0.
0000000000000  \0  \0  \0  \0  \0  \0 001 032  \0  \0  \0  \0  \0  \0  \0   /
0000000000016  \0  \0  \0  \0  \0  \0  \0 032  \0  \0  \0  \0  \0  \0  \0 013
0000000000032  \0  \0  \0  \0  \0  \0 001 021  \0  \0  \0  \0  \0  \0  \0 237
0000000000048   s   u   n   |   s   u   n   2   |   s   u   n   3   |   s   u
0000000000064   n       m   i   c   r   o   s   y   s   t   e   m   s       i
0000000000080   n   c       w   o   r   k   s   t   a   t   i   o   n  \0  \0
0000000000096 001  \0  \0  \0  \0  \0  \0 001  \0  \0  \0  \0 001 001  \0  \0
0000000000112  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0   q
0000000000128  \0  \0  \0  \0  \0  \0  \0   P 377 377 377 377 377 377 377 377
0000000000144  \0  \0  \0  \0  \0  \0  \0   " 377 377 377 377 377 377 377 377
0000000000160 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
0000000000224  \0  \0  \0  \0  \0  \0  \0   /  \0  \0  \0  \0  \0  \0  \0   3
0000000000240 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
0000000000256  \0  \0  \0  \0  \0  \0  \0   1  \0  \0  \0  \0  \0  \0  \0
0000000000272  \0  \0  \0  \0  \0  \0  \0   Z 377 377 377 377 377 377 377 377
0000000000288 377 377 377 377 377 377 377 377  \0  \0  \0  \0  \0  \0  \0   =
```

```
0000000000304  \0  \0  \0  \0  \0  \0  \0   7 377 377 377 377 377 377 377 377
0000000000320 377 377 377 377 377 377 377 377  \0  \0  \0  \0  \0  \0  \0   5
0000000000336 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
0000000000352  \0  \0  \0  \0  \0  \0  \0   9 377 377 377 377 377 377 377 377
0000000000368  \0  \0  \0  \0  \0  \0  \0   N 377 377 377 377 377 377 377 377
0000000000384  \0  \0  \0  \0  \0  \0  \0   R  \0  \0  \0  \0  \0  \0  \0   V
0000000000400 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
0000000000496  \0  \0  \0  \0  \0  \0  \0 232 377 377 377 377 377 377 377 377
0000000000512 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
0000000000560  \0  \0  \0  \0  \0  \0  \0 222 377 377 377 377 377 377 377 377
0000000000576 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
0000000000624 377 377 377 377 377 377 377 377  \0  \0  \0  \0  \0  \0  \0   d
0000000000640  \0  \0  \0  \0  \0  \0  \0   h 377 377 377 377 377 377 377 377
0000000000656 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
0000000000704  \0  \0  \0  \0  \0  \0  \0   r 377 377 377 377 377 377 377 377
0000000000720 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
0000000000736 377 377 377 377 377 377 377 377  \0  \0  \0  \0  \0  \0  \0   ~
0000000000752 377 377 377 377 377 377 377 377  \0  \0  \0  \0  \0  \0  \0 202
0000000000768  \0  \0  \0  \0  \0  \0  \0 206  \0  \0  \0  \0  \0  \0  \0 212
0000000000784 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
0000000000816 377 377 377 377 377 377 377 377  \0  \0  \0  \0  \0  \0  \0 216
0000000000832 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
0000000000848  \0  \0  \0  \0  \0  \0  \0   n 377 377 377 377 377 377 377 377
0000000000864 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
0000000000880  \0  \0  \0  \0  \0  \0  \0   v 377 377 377 377 377 377 377 377
0000000000896 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
0000000000912  \0  \0  \0  \0  \0  \0  \0   z 377 377 377 377 377 377 377 377
0000000000928 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
0000000001200  \0  \0  \0  \0  \0  \0  \0 226 377 377 377 377 377 377 377 377
0000000001216 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
0000000001248  \0  \0  \0  \0  \0  \0  \0   l 377 377 377 377 377 377 377 377
0000000001264 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
0000000001280 377 377 377 377 377 377 377 377  \0  \0  \0  \0  \0  \0  \0   b
0000000001296 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377 377
*
0000000002400   s   u   n   |   s   u   n   2   |   s   u   n   3   |   s   u
0000000002416   n       m   i   c   r   o   s   y   s   t   e   m   s       i
```

```
0000000002432   n   c       w   o   r   k   s   t   a   t   i   o   n  \0 007
0000000002448  \0  \f  \0  \r  \0  \b  \0  \n  \0 033   [   C  \0 033   [   %
0000000002464   i   %   p   1   %   d   ;   %   p   2   %   d   H  \0 033   [
0000000002480   A  \0 033   [   P  \0 033   [   M  \0 033   [   J  \0 033   [
0000000002496   K  \0  \t  \0 033   [   @  \0 033   [   L  \0  \n  \0 033   [
0000000002512   D  \0 033   [   B  \0 033   [   C  \0 033   [   A  \0 033   O
0000000002528   P  \0 033   O   Q  \0 033   O   R  \0 033   O   S  \0 033   [
0000000002544   H  \0 033   [   m  \0 033   [   s  \0 033   [   7   m  \0  \0
0000000002559
```

FILES

`/usr/include/term.h`	`terminfo`(5) header file
`/usr/lib/terminfo/?/*`	Compiled terminal description database

SEE ALSO

`terminfo`(5)

`curses`(3) (available only online)

`term`(7) (available only online)

`infocmp`(8), `tic`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

terminfo – Terminal capability database

SYNOPSIS

/usr/lib/terminfo/?/*

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The terminfo file format is a compiled database (see tic(8)) that describes the capabilities of terminals. Terminals are described in terminfo source descriptions by giving a set of capabilities which they have, by describing how operations are performed, by describing padding requirements, and by specifying initialization sequences. This database is used by applications programs, such as vi(1) and curses(3), so they can work with a variety of terminals without changes to the programs. To obtain the source description for a terminal, use the -I option of infocmp(8).

Entries in terminfo source files consist of several comma-separated fields. White space after each comma is ignored. The first line of each terminal description in the terminfo database gives the name by which terminfo knows the terminal, separated by | symbols. The first name given is the most common abbreviation for the terminal (this is the one to use to set the TERM environment variable in $HOME/.profile; see profile(5)), the last name given should be a long name that fully identifies the terminal, and all others are understood as synonyms for the terminal name. All names but the last should contain no blanks and must be unique in the first 14 characters; the last name may contain blanks for readability.

You should select terminal names (except for the last, verbose entry) by using the following conventions. The particular piece of hardware making up the terminal should have a root name chosen (for example, for the AT&T 4425 terminal, att4425). To indicate modes in which the hardware can be, or user preferences, append a hyphen and an indicator of the mode. For examples and more information on choosing names and synonyms, see term(5).

Capabilities

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

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

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

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

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

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

In the following table, the Name column lists the capname and the Code column lists the termcap code.

Variable	Name	Code	Description
Booleans:			
`auto_left_margin`	`bw`	`bw`	`cub1` wraps from column 0 to last column.
`auto_right_margin`	`am`	`am`	Terminal has automatic margin.
`no_esc_ctlc`	`xsb`	`xb`	Beehive (f1=< ESCAPE/ >, f2=< CONTROL-C/ >).
`ceol_standout_glitch`	`xhp`	`xs`	Standout not erased by overwriting (hp).
`eat_newline_glitch`	`xenl`	`xn`	New line ignored after 80 columns (Concept).
`erase_overstrike`	`eo`	`eo`	Can erase overstrikes with a blank.
`generic_type`	`gn`	`gn`	Generic line type (for example, dialup, switch).
`hard_copy`	`hc`	`hc`	Hard-copy terminal.
`hard_cursor`	`chts`	`HC`	Cursor is hard to see.
`has_meta_key`	`km`	`km`	Has a meta key (shift, sets parity bit).
`has_status_line`	`hs`	`hs`	Has extra "status line."
`insert_null_glitch`	`in`	`in`	Insert mode distinguishes nulls.
`memory_above`	`da`	`da`	Display may be retained above the screen.
`memory_below`	`db`	`db`	Display may be retained below the screen.
`move_insert_mode`	`mir`	`mi`	Safe to move while in insert mode.
`move_standout_mode`	`msgr`	`ms`	Safe to move in standout modes.
`needs_xon_xoff`	`nxon`	`nx`	Padding will not work, `xon`/`xoff` required.
`non_rev_rmcup`	`nrrmc`	`NR`	`smcup` does not reverse `rmcup`.
`no_pad_char`	`npc`	`NP`	Pad character does not exist.
`over_strike`	`os`	`os`	Terminal overstrikes on hard-copy terminal.
`prtr_silent`	`mc5i`	`5i`	Printer does not echo on screen.
`status_line_esc_ok`	`eslok`	`es`	Escape can be used on the status line.
`dest_tabs_magic_smso`	`xt`	`xt`	Destructive tabs, magic `smso` character (t1061).
`tilde_glitch`	`hz`	`hz`	Hazeltine; cannot print tildes(~).
`transparent_underline`	`ul`	`ul`	Underline character overstrikes.
`xon_xoff`	`xon`	`xo`	Terminal uses `xon`/`xoff` handshaking.
Numbers:			
`columns`	`cols`	`co`	Number of columns in a line.

Variable	Name	Code	Description
init_tabs	it	it	Tabs initially every # spaces.
label_height	lh	lh	Number of rows in each label.
label_width	lw	lw	Number of columns in each label.
lines	lines	li	Number of lines on screen or page.
lines_of_memory	lm	lm	Lines of memory if > lines; 0 means varies.
magic_cookie_glitch	xmc	sg	Number blank characters left by smso or rmso.
num_labels	nlab	Nl	Number of labels on screen (start at 1).
padding_baud_rate	pb	pb	Lowest baud rate where padding needed.
virtual_terminal	vt	vt	Virtual terminal number (UNIX system).
width_status_line	wsl	ws	Number of columns in status line.
Strings:			
acs_chars	acsc	ac	Graphic charset pairs aAbBcC – def=vt100+.
back_tab	cbt	bt	Back tab.
bell	bel	bl	Audible signal (bell).
carriage_return	cr	cr	Carriage return (∗).
change_scroll_region	csr	cs	Change to lines #1 through #2 (vt100) (G).
char_padding	rmp	rP	Like ip but when in replace mode.
clear_all_tabs	tbc	ct	Clear all tab stops.
clear_margins	mgc	MC	Clear left and right soft margins.
clear_screen	clear	cl	Clear screen and home cursor (∗).
clr_bol	el1	cb	Clear to beginning-of-line, inclusive.
clr_eol	el	ce	Clear to end-of-line.
clr_eos	ed	cd	Clear to end-of-display (∗).
column_address	hpa	ch	Horizontal position absolute (G).
command_character	cmdch	CC	Term. settable cmd char in prototype.
cursor_address	cup	cm	Cursor motion to row #1 col #2 (G).
cursor_down	cud1	do	Down 1 line.
cursor_home	home	ho	Home cursor (if no cup).
cursor_invisible	civis	vi	Make cursor invisible.
cursor_left	cub1	le	Move cursor left one space.
cursor_mem_address	mrcup	CM	Memory relative cursor addressing (G).
cursor_normal	cnorm	ve	Make cursor appear normal (undo vs/vi).
cursor_right	cuf1	nd	Nondestructive space (cursor right).
cursor_to_ll	ll	ll	Last line, first column (if no cup).
cursor_up	cuu1	up	Upline (cursor up).
cursor_visible	cvvis	vs	Make cursor very visible.
delete_character	dch1	dc	Delete character (∗).
delete_line	dl1	dl	Delete line (∗).
dis_status_line	dsl	ds	Disable status line.

Variable	Name	Code	Description
down_half_line	hd	hd	Half-line down (forward 1/2 line feed).
ena_acs	enacs	eA	Enable alternate character set.
enter_alt_charset_mode	smacs	as	Start alternate character set.
enter_am_mode	smam	SA	Turn on automatic margins.
enter_blink_mode	blink	mb	Turn on blinking.
enter_bold_mode	bold	md	Turn on bold (extra bright) mode.
enter_ca_mode	smcup	ti	String to begin programs that use cup.
enter_delete_mode	smdc	dm	Delete mode (enter).
enter_dim_mode	dim	mh	Turn on half-bright mode.
enter_insert_mode	smir	im	Insert mode (enter).
enter_protected_mode	prot	mp	Turn on protected mode.
enter_reverse_mode	rev	mr	Turn on reverse video mode.
enter_secure_mode	invis	mk	Turn on blank mode (characters invisible).
enter_standout_mode	smso	so	Begin standout mode.
enter_underline_mode	smul	us	Start underscore mode.
enter_xon_mode	smxon	SX	Turn on xon/xoff handshaking.
erase_chars	ech	ec	Erase #1 characters (G).
exit_alt_charset_mode	rmacs	ae	End alternate character set.
exit_am_mode	rmam	RA	Turn off automatic margins.
exit_attribute_mode	sgr0	me	Turn off all attributes.
exit_ca_mode	rmcup	te	String to end programs that use cup.
exit_delete_mode	rmdc	ed	End delete mode.
exit_insert_mode	rmir	ei	End insert mode.
exit_standout_mode	rmso	se	End standout mode.
exit_underline_mode	rmul	ue	End underscore mode.
exit_xon_mode	rmxon	RX	Turn off xon/xoff handshaking.
flash_screen	flash	vb	Visible bell (may not move cursor).
form_feed	ff	ff	Hard-copy terminal page eject (*).
from_status_line	fsl	fs	Return from status line.
init_1string	is1	i1	Terminal initialization string.
init_2string	is2	is	Terminal initialization string.
init_3string	is3	i3	Terminal initialization string.
init_file	if	if	Name of initialization file that contains is.
init_prog	iprog	iP	Path name of program for init.
insert_character	ich1	ic	Insert character.
insert_line	il1	al	Add new blank line (*).
insert_padding	ip	ip	Insert pad after character inserted (*).
key_a1	ka1	K1	KEY_A1, 0534, Upper left of keypad.
key_a3	ka3	K3	KEY_A3, 0535, Upper right of keypad.
key_b2	kb2	K2	KEY_B2, 0536, Center of keypad.

Variable	Name	Code	Description
key_backspace	kbs	kb	KEY_BACKSPACE, 0407, Sent by backspace key.
key_beg	kbeg	@1	KEY_BEG, 0542, Sent by beg(inning) key.
key_btab	kcbt	kB	KEY_BTAB, 0541, Sent by back-tab key.
key_c1	kc1	K4	KEY_C1, 0537, Lower left of keypad.
key_c3	kc3	K5	KEY_C3, 0540, Lower right of keypad.
key_cancel	kcan	@2	KEY_CANCEL, 0543, Sent by cancel key.
key_catab	ktbc	ka	KEY_CATAB, 0526, Sent by clear-all-tabs key.
key_clear	kclr	kC	KEY_CLEAR, 0515, Sent by clear-screen or erase key.
key_close	kclo	@3	KEY_CLOSE, 0544, Sent by close key.
key_command	kcmd	@4	KEY_COMMAND, 0545, Sent by cmd (command) key.
key_copy	kcpy	@5	KEY_COPY, 0546, Sent by copy key.
key_create	kcrt	@6	KEY_CREATE, 0547, Sent by create key.
key_ctab	kctab	kt	KEY_CTAB, 0525, Sent by clear-tab key.
key_dc	kdch1	kD	KEY_DC, 0512, Sent by delete-character key.
key_dl	kdl1	kL	KEY_DL, 0510, Sent by delete-line key.
key_down	kcud1	kd	KEY_DOWN, 0402, Sent by terminal down-arrow key.
key_eic	krmir	kM	KEY_EIC, 0514, Sent by rmir or smir in insert mode.
key_end	kend	@7	KEY_END, 0550, Sent by end key.
key_enter	kent	@8	KEY_ENTER, 0527, Sent by enter/send key.
key_eol	kel	kE	KEY_EOL, 0517, Sent by clear-to-end-of-line key.
key_eos	ked	kS	KEY_EOS, 0516, Sent by clear-to-end-of-screen key.
key_exit	kext	@9	KEY_EXIT, 0551, Sent by exit key.
key_f0	kf0	k0	KEY_F(0), 0410, Sent by function key f0.
key_f1	kf1	k1	KEY_F(1), 0411, Sent by function key f1.
key_f2	kf2	k2	KEY_F(2), 0412, Sent by function key f2.
key_f3	kf3	k3	KEY_F(3), 0413, Sent by function key f3.
key_f4	kf4	k4	KEY_F(4), 0414, Sent by function key f4.
key_f5	kf5	k5	KEY_F(5), 0415, Sent by function key f5.
key_f6	kf6	k6	KEY_F(6), 0416, Sent by function key f6.
key_f7	kf7	k7	KEY_F(7), 0417, Sent by function key f7.
key_f8	kf8	k8	KEY_F(8), 0420, Sent by function key f8.
key_f9	kf9	k9	KEY_F(9), 0421, Sent by function key f9.
key_f10	kf10	k;	KEY_F(10), 0422, Sent by function key f10.

Variable	Name	Code	Description
key_f11	kf11	F1	KEY_F(11), 0423, Sent by function key f11.
key_f12	kf12	F2	KEY_F(12), 0424, Sent by function key f12.
key_f13	kf13	F3	KEY_F(13), 0425, Sent by function key f13.
key_f14	kf14	F4	KEY_F(14), 0426, Sent by function key f14.
key_f15	kf15	F5	KEY_F(15), 0427, Sent by function key f15.
key_f16	kf16	F6	KEY_F(16), 0430, Sent by function key f16.
key_f17	kf17	F7	KEY_F(17), 0431, Sent by function key f17.
key_f18	kf18	F8	KEY_F(18), 0432, Sent by function key f18.
key_f19	kf19	F9	KEY_F(19), 0433, Sent by function key f19.
key_f20	kf20	FA	KEY_F(20), 0434, Sent by function key f20.
key_f21	kf21	FB	KEY_F(21), 0435, Sent by function key f21.
key_f22	kf22	FC	KEY_F(22), 0436, Sent by function key f22.
key_f23	kf23	FD	KEY_F(23), 0437, Sent by function key f23.
key_f24	kf24	FE	KEY_F(24), 0440, Sent by function key f24.
key_f25	kf25	FF	KEY_F(25), 0441, Sent by function key f25.
key_f26	kf26	FG	KEY_F(26), 0442, Sent by function key f26.
key_f27	kf27	FH	KEY_F(27), 0443, Sent by function key f27.
key_f28	kf28	FI	KEY_F(28), 0444, Sent by function key f28.
key_f29	kf29	FJ	KEY_F(29), 0445, Sent by function key f29.
key_f30	kf30	FK	KEY_F(30), 0446, Sent by function key f30.
key_f31	kf31	FL	KEY_F(31), 0447, Sent by function key f31.
key_f32	kf32	FM	KEY_F(32), 0450, Sent by function key f32.
key_f33	kf33	FN	KEY_F(13), 0451, Sent by function key f13.
key_f34	kf34	FO	KEY_F(34), 0452, Sent by function key f34.
key_f35	kf35	FP	KEY_F(35), 0453, Sent by function key f35.
key_f36	kf36	FQ	KEY_F(36), 0454, Sent by function key f36.
key_f37	kf37	FR	KEY_F(37), 0455, Sent by function key f37.
key_f38	kf38	FS	KEY_F(38), 0456, Sent by function key f38.
key_f39	kf39	FT	KEY_F(39), 0457, Sent by function key f39.
key_f40	kf40	FU	KEY_F(40), 0460, Sent by function key f40.
key_f41	kf41	FV	KEY_F(41), 0461, Sent by function key f41.
key_f42	kf42	FW	KEY_F(42), 0462, Sent by function key f42.
key_f43	kf43	FX	KEY_F(43), 0463, Sent by function key f43.
key_f44	kf44	FY	KEY_F(44), 0464, Sent by function key f44.
key_f45	kf45	FZ	KEY_F(45), 0465, Sent by function key f45.
key_f46	kf46	Fa	KEY_F(46), 0466, Sent by function key f46.
key_f47	kf47	Fb	KEY_F(47), 0467, Sent by function key f47.
key_f48	kf48	Fc	KEY_F(48), 0470, Sent by function key f48.
key_f49	kf49	Fd	KEY_F(49), 0471, Sent by function key f49.
key_f50	kf50	Fe	KEY_F(50), 0472, Sent by function key f50.

Variable	Name	Code	Description
key_f51	kf51	Ff	KEY_F(51), 0473, Sent by function key f51.
key_f52	kf52	Fg	KEY_F(52), 0474, Sent by function key f52.
key_f53	kf53	Fh	KEY_F(53), 0475, Sent by function key f53.
key_f54	kf54	Fi	KEY_F(54), 0476, Sent by function key f54.
key_f55	kf55	Fj	KEY_F(55), 0477, Sent by function key f55.
key_f56	kf56	Fk	KEY_F(56), 0500, Sent by function key f56.
key_f57	kf57	Fl	KEY_F(57), 0501, Sent by function key f57.
key_f58	kf58	Fm	KEY_F(58), 0502, Sent by function key f58.
key_f59	kf59	Fn	KEY_F(59), 0503, Sent by function key f59.
key_f60	kf60	Fo	KEY_F(60), 0504, Sent by function key f60.
key_f61	kf61	Fp	KEY_F(61), 0505, Sent by function key f61.
key_f62	kf62	Fq	KEY_F(62), 0506, Sent by function key f62.
key_f63	kf63	Fr	KEY_F(63), 0507, Sent by function key f63.
key_find	kfnd	@0	KEY_FIND, 0552, Sent by find key.
key_help	khlp	%1	KEY_HELP, 0553, Sent by help key.
key_home	khome	kh	KEY_HOME, 0406, Sent by home key.
key_ic	kich1	kI	KEY_IC, 0513, Sent by ins-char/enter ins-mode key.
key_il	kil1	kA	KEY_IL, 0511, Sent by insert-line key.
key_left	kcub1	kl	KEY_LEFT, 0404, Sent by terminal left-arrow key.
key_ll	kll	kH	KEY_LL, 0533, Sent by home-down key.
key_mark	kmrk	%2	KEY_MARK, 0554, Sent by mark key.
key_message	kmsg	%3	KEY_MESSAGE, 0555, Sent by message key.
key_move	kmov	%4	KEY_MOVE, 0556, Sent by move key.
key_next	knxt	%5	KEY_NEXT, 0557, Sent by next-object key.
key_npage	knp	kN	KEY_NPAGE, 0522, Sent by next-page key.
key_open	kopn	%6	KEY_OPEN, 0560, Sent by open key.
key_options	kopt	%7	KEY_OPTIONS, 0561, Sent by options key.
key_ppage	kpp	kP	KEY_PPAGE, 0523, Sent by previous-page key.
key_previous	kprv	%8	KEY_PREVIOUS, 0562, Sent by previous-object key.
key_print	kprt	%9	KEY_PRINT, 0532, Sent by print or copy key.
key_redo	krdo	%0	KEY_REDO, 0563, Sent by redo key.
key_reference	kref	&1	KEY_REFERENCE, 0564, Sent by ref(erence) key.
key_refresh	krfr	&2	KEY_REFRESH, 0565, Sent by refresh key.
key_replace	krpl	&3	KEY_REPLACE, 0566, Sent by replace key.
key_restart	krst	&4	KEY_RESTART, 0567, Sent by restart key.
key_resume	kres	&5	KEY_RESUME, 0570, Sent by resume key.

Variable	Name	Code	Description
key_right	kcuf1	kr	KEY_RIGHT, 0405, Sent by terminal right-arrow key.
key_save	ksav	&6	KEY_SAVE, 0571, Sent by save key.
key_sbeg	kBEG	&9	KEY_SBEG, 0572, Sent by shifted beginning key.
key_scancel	kCAN	&0	KEY_SCANCEL, 0573, Sent by shifted cancel key.
key_scommand	kCMD	*1	KEY_SCOMMAND, 0574, Sent by shifted command key.
key_scopy	kCPY	*2	KEY_SCOPY, 0575, Sent by shifted copy key.
key_screate	kCRT	*3	KEY_SCREATE, 0576, Sent by shifted create key.
key_sdc	kDC	*4	KEY_SDC, 0577, Sent by shifted delete-char key.
key_sdl	kDL	*5	KEY_SDL, 0600, Sent by shifted delete-line key.
key_select	kslt	*6	KEY_SELECT, 0601, Sent by select key.
key_send	kEND	*7	KEY_SEND, 0602, Sent by shifted end key.
key_seol	kEOL	*8	KEY_SEOL, 0603, Sent by shifted clear-line key.
key_sexit	kEXT	*9	KEY_SEXIT, 0604, Sent by shifted exit key.
key_sf	kind	kF	KEY_SF, 0520, Sent by scroll-forward/down key.
key_sfind	kFND	*0	KEY_SFIND, 0605, Sent by shifted find key.
key_shelp	kHLP	#1	KEY_SHELP, 0606, Sent by shifted help key.
key_shome	kHOM	#2	KEY_SHOME, 0607, Sent by shifted home key.
key_sic	kIC	#3	KEY_SIC, 0610, Sent by shifted input key.
key_sleft	kLFT	#4	KEY_SLEFT, 0611, Sent by shifted left-arrow key.
key_smessage	kMSG	%a	KEY_SMESSAGE, 0612, Sent by shifted message key.
key_smove	kMOV	%b	KEY_SMOVE, 0613, Sent by shifted move key.
key_snext	kNXT	%c	KEY_SNEXT, 0614, Sent by shifted next key.
key_soptions	kOPT	%d	KEY_SOPTIONS, 0615, Sent by shifted options key.
key_sprevious	kPRV	%e	KEY_SPREVIOUS, 0616, Sent by shifted prev key.
key_sprint	kPRT	%f	KEY_SPRINT, 0617, Sent by shifted print key.
key_sr	kri	kR	KEY_SR, 0521, Sent by scroll-backward/up key.
key_sredo	kRDO	%g	KEY_SREDO, 0620, Sent by shifted redo key.
key_sreplace	kRPL	%h	KEY_SREPLACE, 0621, Sent by shifted replace key.
key_sright	kRIT	%i	KEY_SRIGHT, 0622, Sent by shifted right-arrow key.

Variable	Name	Code	Description
key_srsume	kRES	%j	KEY_SRSUME, 0623, Sent by shifted resume key.
key_ssave	kSAV	!1	KEY_SSAVE, 0624, Sent by shifted save key.
key_ssuspend	kSPD	!2	KEY_SSUSPEND, 0625, Sent by shifted suspend key.
key_stab	khts	kT	KEY_STAB, 0524, Sent by set-tab key.
key_sundo	kUND	!3	KEY_SUNDO, 0626, Sent by shifted undo key.
key_suspend	kspd	&7	KEY_SUSPEND, 0627, Sent by suspend key.
key_undo	kund	&8	KEY_UNDO, 0630, Sent by undo key.
key_up	kcuu1	ku	KEY_UP, 0403, Sent by terminal up-arrow key.
keypad_local	rmkx	ke	Out of ‘‘keypad-transmit’’ mode.
keypad_xmit	smkx	ks	Put terminal in ‘‘keypad-transmit’’ mode.
lab_f0	lf0	l0	Labels on function key f0 if not f0.
lab_f1	lf1	l1	Labels on function key f1 if not f1.
lab_f2	lf2	l2	Labels on function key f2 if not f2.
lab_f3	lf3	l3	Labels on function key f3 if not f3.
lab_f4	lf4	l4	Labels on function key f4 if not f4.
lab_f5	lf5	l5	Labels on function key f5 if not f5.
lab_f6	lf6	l6	Labels on function key f6 if not f6.
lab_f7	lf7	l7	Labels on function key f7 if not f7.
lab_f8	lf8	l8	Labels on function key f8 if not f8.
lab_f9	lf9	l9	Labels on function key f9 if not f9.
lab_f10	lf10	la	Labels on function key f10 if not f10.
label_off	rmln	LF	Turn off soft labels.
label_on	smln	LO	Turn on soft labels.
meta_off	rmm	mo	Turn off "meta mode".
meta_on	smm	mm	Turn on "meta mode" (8th bit).
newline	nel	nw	New line (behaves like cr followed by lf).
pad_char	pad	pc	Pad character (rather than null).
parm_dch	dch	DC	Delete #1 chars (G*).
parm_delete_line	dl	DL	Delete #1 lines (G*).
parm_down_cursor	cud	DO	Move cursor down #1 lines. (G*).
parm_ich	ich	IC	Insert #1 blank chars (G*).
parm_index	indn	SF	Scroll forward #1 lines. (G).
parm_insert_line	il	AL	Add #1 new blank lines (G*).
parm_left_cursor	cub	LE	Move cursor left #1 spaces (G).
parm_right_cursor	cuf	RI	Move cursor right #1 spaces. (G*).
parm_rindex	rin	SR	Scroll backward #1 lines. (G).
parm_up_cursor	cuu	UP	Move cursor up #1 lines. (G*).
pkey_key	pfkey	pk	Prog funct key #1 to type string #2.

Variable	Name	Code	Description
pkey_local	pfloc	pl	Prog funct key #1 to execute string #2.
pkey_xmit	pfx	px	Prog funct key #1 to xmit string #2.
plab_norm	pln	pn	Prog label #1 to show string #2.
print_screen	mc0	ps	Print contents of the screen.
prtr_non	mc5p	pO	Turn on the printer for #1 bytes.
prtr_off	mc4	pf	Turn off the printer.
prtr_on	mc5	po	Turn on the printer.
repeat_char	rep	rp	Repeat char #1 #2 times (G*).
req_for_input	rfi	RF	Send next input character (for ptys).
reset_1string	rs1	r1	Reset terminal completely to sane modes.
reset_2string	rs2	r2	Reset terminal completely to sane modes.
reset_3string	rs3	r3	Reset terminal completely to sane modes.
reset_file	rf	rf	Name of file containing reset string.
restore_cursor	rc	rc	Restore cursor to position of last sc
row_address	vpa	cv	Vertical position absolute (G).
save_cursor	sc	sc	Save cursor position.
scroll_forward	ind	sf	Scroll text up.
scroll_reverse	ri	sr	Scroll text down.
set_attributes	sgr	sa	Define the video attributes #1-#9 (G).
set_left_margin	smgl	ML	Set soft left margin.
set_right_margin	smgr	MR	Set soft right margin.
set_tab	hts	st	Set a tab in all rows, current column.
set_window	wind	wi	Current window is lines #1–#2 cols #3–#4 (G).
tab	ht	ta	Tab to next 8 space hardware tab stop.
to_status_line	tsl	ts	Go to status line, col #1 (G).
underline_char	uc	uc	Underscore 1 character and move past it.
up_half_line	hu	hu	Half-line up (reverse 1/2 line-feed).
xoff_character	xoffc	XF	X-off character.
xon_character	xonc	XN	X-on character.

Sample Entry

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

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

To continue entries onto multiple lines, place white space at the beginning of each line except the first. Lines that begin with # are comment lines. Capabilities in `terminfo` are of three types: Boolean capabilities, which indicate that the terminal has some particular feature; numeric capabilities, which give the size of the terminal or particular features; and string capabilities, which give a sequence that can perform particular terminal operations.

Types of Capabilities

All capabilities have names. For instance, the fact that the Concept has *automatic margins* (that is, an automatic return and line feed when the end of a line is reached) is indicated by the capability `am`. Hence, the description of the Concept includes `am`. Numeric capabilities are followed by the # symbol and then the value. Thus, `cols`, which indicates the number of columns the terminal has, gives the value 80 for the Concept. You may specify the value in decimal, octal, or hexadecimal using typical C conventions.

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

Several escape sequences are provided in the string-valued capabilities for easy encoding of characters. Both \E and \e map to an ESCAPE character, ^*x* maps to a control-*x* for any appropriate *x*, and the sequences \n, \l, \r, \t, \b, \f, and \s give a new line, line feed, return, tab, backspace, form feed, and space, respectively. Other escapes include \^ for caret (^); \\ for backslash (\); \, for comma (,); \: for colon (:); and \0 for null. (\0 actually produces \200, which does not terminate a string but it behaves as a null character on most terminals.) Finally, you may specify characters as three octal digits after a backslash (for example, \123).

Sometimes you must comment out individual capabilities by putting a period before the capability name, (for example, see the second ind in the previous example). Capabilities are defined in a left-to-right order; therefore, a prior definition will override a later definition.

Preparing Descriptions

The most effective way to prepare a terminal description is by imitating the description of a similar terminal in terminfo and to build up a description gradually, using partial descriptions with vi(1) to check that they are correct. Be aware that a very unusual terminal may expose deficiencies in the ability of the terminfo file to describe it or the inability of vi(1) to work with that terminal. To test a new terminal description, set the TERMINFO environment variable to a path name of a directory that contains the compiled description on which you are working and programs will look there rather than in /usr/lib/terminfo. To get the padding for insert-line correct (if the terminal manufacturer did not document it), a severe test is to comment out xon, edit a large file at 9600 Bd with vi(1), delete 16 or so lines from the middle of the screen, then hit the <u> key several times quickly. If the display is corrupted, more padding usually is needed. You can use a similar test for insert-character.

Basic Capabilities

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

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

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

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

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

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

These capabilities suffice to describe hard-copy and screen terminals. Thus, the model 33 teletype is described as follows:

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

The Lear Siegler ADM-3 is described as follows:

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

Parameterized Strings

Cursor addressing and other strings that require parameters in the terminal are described by a parameterized string capability, with `printf`(3C)-like escapes (`%x`) in it. For example, to address the cursor, the `cup` capability is given, using two parameters: the row and column to which to address. (Rows and columns are numbered from 0 and refer to the physical screen visible to the user, not to any unseen memory.) If the terminal has memory-relative cursor addressing, you can indicate that by using `mrcup`.

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

The % encodings have the following meanings:

% encoding	**Meaning**
%%	Outputs '%'
%[[:]*flags*][*width*[.*precision*]][doxXs]	As in printf, flags are [-+#] and space
%c	Prints pop() gives %c
%p[1-9]	Pushes i^{th} parameter
%P[a-z]	Pets variable [a-z] to pop()
%g[a-z]	Pets variable [a-z] and pushes it
%'*c*'	Pushes char constant *c*
%{*nn*}	Pushes decimal constant *nn*
%l	Push strlen(pop())
%+ %- %* %/ %m	Arithmetic (%m is mod): push(pop() op pop())
%& %| %^	Bit operations: push(pop() op pop())
%= %> %<	Logical operations: push(pop() op pop())
%A %O	Logical operations: and, or
%! %~	Unary operations: push(op pop())
%i	(for ANSI terminals) Add 1 to first parm, if one parm present, or first two parms, if more than one parm present

`%? expr %t thenpart %e elsepart %;`

If-then-else, %e elsepart is optional; else-if's are possible as in the Algol 68 language:
%? c_1 %t b_1 %e c_2 %t b_2 %e c_3 %t b_3 %e c_4 %t b_4 %e b_5%;
c_i are conditions, b_i are bodies.

If you use the `-` flag with ''`%[doxXs]`'', you must place a `:` between the `%` and the `-` to differentiate the flag from the binary `%-` operator (for example, `%:-16.16s`).

Consider the Hewlett-Packard 2645, which, to get to row 3 and column 12, must be sent `\E&a12c03Y` padded for 6 ms. The order of the rows and columns is inverted here, and that the row and column are zero-padded as 2 digits. Thus, its `cup` capability is ''`cup=\E&a%p2%2.2dc%p1%2.2dY$<6>`''.

The Micro-Term ACT-IV needs the current row and column sent preceded by a `^T`, with the row and column simply encoded in binary, `cup=^T%p1%c%p2%c`. Terminals that use `%c` must be able to backspace the cursor (`cub1`), and to move the cursor up one line on the screen (`cuu1`). This is necessary because it is not always safe to transmit `\n`, `^D`, and `\r`, because the system may change or discard them. (The library routines that deal with `terminfo` set tty modes so that tabs are never expanded; therefore, `\t` is safe to send. This turns out to be essential for the Ann Arbor 4080.)

A final example is the LSI ADM-3a, which uses row and column offset by a blank character; thus, ''`cup=\E=%p1%'\s'%+%c%p2%'\s'%+%c`''. After sending ''`\E=`'', this pushes the first parameter, pushes the ASCII value for a space (32), adds them (pushing the sum on the stack in place of the two previous values), and outputs that value as a character. Then the same is done for the second parameter. You can do more complex arithmetic by using the stack.

Cursor Motions

If the terminal has a fast way to home the cursor (to very upper left corner of screen), you can specify this as `home`; similarly, a fast way of getting to the lower left corner can be given as `ll`; this may involve going up with `cuu1` from the home position, but a program should never do this itself (unless `ll` does), because it cannot make assumptions about the effect of moving up from the home position. The home position is the same as addressing to (0,0): to the top left corner of the screen, not of memory. (Thus, you cannot use the `\EH` sequence on Hewlett-Packard terminals for `home` without losing some of the other features on the terminal.)

If the terminal has row or column absolute-cursor addressing, you can specify these as single-parameter capabilities `hpa` (horizontal position absolute) and `vpa` (vertical position absolute). Sometimes these are shorter than the more general two-parameter sequence (as with the Hewlett-Packard 2645) and can be used in preference to `cup`. If parameterized local motions (for example, move *n* spaces to the right) exist, you can specify these as `cud`, `cub`, `cuf`, and `cuu` with one parameter that indicates how many spaces to move. These are primarily useful if the terminal does not have `cup`, such as the Tektronix 4025.

Area Clears

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

Insert/Delete Line

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

If the terminal has a settable destructive scrolling region (such as the VT100), you can describe the command to set this by using the `csr` capability, which takes two parameters: the top and bottom lines of the scrolling region. The cursor position is undefined after using this command. You can get the effect of insert or delete line by using this command; the `sc` and `rc` (save and restore cursor) commands also are useful. You also can insert lines at the top or bottom of the screen by using `ri` or `ind` on many terminals without a true insert/delete line, and it is often faster even on terminals that have those features.

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

If the terminal can define a window as part of memory, which all commands affect, you should specify it as the parameterized string `wind`. The four parameters are the starting and ending lines in memory and the starting and ending columns in memory, in that order.

If the terminal can retain display memory above, you should specify `da` capability; if display memory can be retained below, you should specify `db`. These indicate that deleting a line or scrolling a full screen may bring nonblank lines up from below or that scrolling back with `ri` may bring down nonblank lines.

Insert/Delete Character

You can describe two basic kinds of intelligent terminals with respect to insert/delete character operations by using `terminfo`. The most common insert/delete character operations affect only the characters on the current line and shift characters off the end of the line rigidly. Other terminals, such as the Concept 100 and the Perkin Elmer Owl, make a distinction between typed and untyped blanks on the screen, shifting on an insert or delete only to an untyped blank on the screen, which is either eliminated or expanded to two untyped blanks. To determine the kind of terminal you have, clear the screen and then type text separated by cursor motions. Type ''`abc def`'' by using local cursor motions (not spaces) between the `abc` and the `def`. Then position the cursor before the `abc`, and put the terminal in insert mode. If typing characters causes the rest of the line to shift rigidly and characters to fall off the end, your terminal does not distinguish between blanks and untyped positions. If the `abc` shifts over to the `def`, which then move together around the end of the current line and onto the next as you insert, you have the second type of terminal and should give the capability `in`, which stands for ''insert null.'' Although these are two logically-separate attributes (one line versus multiline insert mode and special treatment of untyped spaces), we have seen no terminals whose insert mode cannot be described with one attribute.

The `terminfo` file can describe both terminals that have an insert mode and terminals that send a simple sequence to open a blank position on the current line. To get into insert mode, give `smir` as the sequence. To leave insert mode, give `rmir` as the sequence. Now give as `ich1` any sequence needed to be sent just before sending the character to be inserted. Most terminals with a true insert mode will not give `ich1`; terminals that send a sequence to open a screen position should give it here. (If your terminal has both, insert mode usually is preferable to `ich1`. Do not specify both unless the terminal actually requires both to be used in combination.) If post-insert padding is needed, give this as a number of milliseconds padding in `ip` (a string option). You also may specify any other sequence that may have to be sent after an insert of a single character in `ip`. If your terminal must be placed both into an 'insert mode' and have a special code to precede each inserted character, you can specify both `smir`/`rmir` and `ich1`, and both will be used. The `ich` capability, with one parameter, *n*, repeats the effects of `ich1` *n* times.

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

Occasionally, you may have to move around while in insert mode to delete characters on the same line (for example, if a tab is after the insertion position). If your terminal allows motion while in insert mode, you can specify the `mir` capability to speed up inserting in this case. Omitting `mir` affects only speed. Some terminals (notably Datamedia) must not have `mir` because of the way their insert mode works.

Finally, to delete one character, specify `dch1`. To delete *n* characters, specify `dch` with one argument, *n*. To enter and exit delete mode (any mode the terminal must be placed in for `dch1` to work), specify `smdc` and `rmdc`.

To erase *n* characters (equivalent to outputting *n* blanks without moving the cursor), specify as `ech` with one parameter.

Highlighting, Underlining, and Visible Bells

If your terminal has one or more kinds of display attributes, these can be represented in several different ways. You should choose one display form as *standout mode* (see `curses`(3)), representing a good, high contrast, easy-on-the-eyes format for highlighting error messages and other attention-getters. (If you have a choice, reverse-video plus half-bright is good, or reverse-video alone; however, different users have different preferences on different terminals.) To enter and exit standout mode, specify the sequences `smso` and `rmso`, respectively. If the code to change into or out of standout mode leaves one or even two blank spaces on the screen, as the TVI 912 and Teleray1061 do, you should specify `xmc` to tell how many spaces are left.

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

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

If a sequence exists to set arbitrary combinations of modes, you should specify this as `sgr` (set attributes), taking nine parameters. Each parameter is either 0 or nonzero, because the corresponding attribute is on or off. The nine parameters are, in order, standout, underline, reverse, blink, dim, bold, blank, protect, and alternate character set. Not all modes must be supported by `sgr`; only those for which corresponding separate attribute commands exist. (See the example at the end of this section.)

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

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

If the cursor must be made more visible than normal when it is not on the bottom line (for example, to make a nonblinking underline into an easier-to-find block or blinking underline), give this sequence as `cvvis`. You also should specify the Boolean `chts`. If you can make the cursor completely invisible, specify that as `civis`. You should specify the `cnorm` capability, which undoes the effects of either of these modes.

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

If your terminal generates underlined characters by using the underline character (with no special codes needed) even though it does not otherwise overstrike characters, you should specify the `ul` capability. For terminals in which a character overstriking another leaves both characters on the screen, specify the `os` capability. If overstrikes are erasable with a blank, you should indicate this by specifying `eo`.

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

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

Each escape sequence requires a 0 to turn off other modes before turning on its own mode. As previously suggested, `standout` also is set up to be the combination of `reverse` and `dim`. Because this terminal has no `bold` mode, `bold` is set up as the combination of `reverse` and `underline`. In addition, to allow combinations, such as `underline+blink`, you would use the `\E[0;3;5m` sequence. The terminal does not have `protect` mode either, but that cannot be simulated in any way; therefore, `p8` is ignored. The `altcharset` mode is different in that it is either `^O` or `^N`, depending on whether it is off or on. If all modes were to be turned on, the sequence would be `\E[0;3;4;5;7;8m^N`.

Now look at when different sequences are output (for example, `;3` is output when either `p2` or `p6` is true; that is, if either `underline` or `bold` modes are turned on). Writing out the previous sequences, along with their dependencies, gives the following:

Sequence	When to output	terminfo translation
\E[0	Always	\E[0
;3	If p2 or p6	%?%p2%p6%|%t;3%;
;4	If p1 or p3 or p6	%?%p1%p3%|%p6%|%t;4%;
;5	If p4	%?%p4%t;5%;
;7	If p1 or p5	%?%p1%p5%|%t;7%;
;8	If p7	%?%p7%t;8%;
m	Always	m
^N or ^O	If p9 ^N, else ^O	%?%p9%t^N%e^O%;

Putting this all together into the sgr sequence gives the following:

```
sgr=\E[0%?%p2%p6%|%t;3%;%?%p1%p3%|%p6%|%t;4%;%?%p5%t;5%;%?%p1%
    p5%|%t;7%;%?%p7%t;8%;m%?%p9%t^N%e^O%;,
```

Keypad

If the terminal has a keypad that transmits codes when the keys are pressed, this information can be given. You cannot handle terminals in which the keypad works only in local (this applies, for example, to the unshifted Hewlett-Packard 2621 keys). If the keypad can be set to transmit or not transmit, specify these codes as smkx and rmkx; otherwise, the keypad is assumed to always transmit.

You can specify the codes sent by the left arrow, right arrow, up arrow, down arrow, and home keys as kcub1, kcuf1, kcuu1, kcud1, and khome, respectively. If there are function keys such as f0, f1, . . ., f63, you can specify the codes they send as kf0, kf1, . . ., kf63. If the first 11 keys have labels other than the default f0 through f10, you can specify the labels as lf0, lf1, . . ., lf10. You can specify the codes transmitted by certain other special keys as kll (home down), kbs (backspace), ktbc (clear all tabs), kctab (clear the tab stop in this column), kclr (clear screen or erase key), kdch1 (delete character), kdl1 (delete line), krmir (exit insert mode), kel (clear to end of line), ked (clear to end of screen), kich1 (insert character or enter insert mode), kil1 (insert line), knp (next page), kpp (previous page), kind (scroll forward/down), kri (scroll backward/up), and khts (set a tab stop in this column). If the keypad also has a 3-by-3 array of keys, including the four arrow keys, you can specify the other five keys as ka1, ka3, kb2, kc1, and kc3. These keys are useful when the effects of a 3-by-3 directional pad are needed. Further keys are defined in the previous capabilities list.

You can specify strings to program function keys as pfkey, pfloc, and pfx. You can specify a string to program their soft-screen labels as pln. Each of these strings takes two parameters: the function key number to program (from 0 to 10) and the string with which to program it. Function key numbers out of this range may program undefined keys in a terminal-dependent manner. The difference between the capabilities is that pfkey causes pressing the given key to be the same as the user typing the given string; pfloc causes the string to be executed by the terminal in local mode; and pfx causes the string to be transmitted to the computer. The nlab, lw, and lh capabilities define how many soft labels there are and their width and height. If commands exist to turn the labels on and off, specify them in smln and rmln. Usually, smln is output after one or more pln sequences to make sure that the change becomes visible.

Tabs and Initialization

If the terminal has hardware tabs, you can specify the command to advance to the next tab stop as `ht` (usually control I). You can specify a ''backtab'' command that moves leftward to the next tab stop as `cbt`. By convention, if the teletype modes indicate that tabs are being expanded by the computer rather than being sent to the terminal, programs should not use `ht` or `cbt` even if they are present, because the user may not have set the tab stops properly. If the terminal has hardware tabs that are initially set every *n* spaces when the terminal is powered up, the numeric parameter `it` is given, showing the number of spaces to which the tabs are set. Usually, `tput init` (see `tput`(1)) uses this to determine whether to set the mode for hardware tab expansion and whether to set the tab stops. If the terminal has tab stops that can be saved in nonvolatile memory, the `terminfo` description can assume that they are set properly. If there are commands to set and clear tab stops, you can specify them as `tbc` (clear all tab stops) and `hts` (set a tab stop in the current column of every row).

Other capabilities include `is1`, `is2`, and `is3` initialization strings for the terminal; `iprog`, the path name of a program to be run to initialize the terminal; and `if`, the name of a file that contains long initialization strings. These strings are expected to set the terminal into modes consistent with the rest of the `terminfo` description. They must be sent to the terminal each time the user logs in and be output in the following order: run the program `iprog`; output `is1`; output `is2`; set the margins by using `mgc`, `smgl`, and `smgr`; set the tabs by using `tbc` and `hts`; print the file `if`; and finally output `is3`. Usually, you can do this by using the `init` option of `tput`(1); see `profile`(5).

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

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

If there are commands to set and clear margins, you can specify them as `mgc` (clear all margins), `smgl` (set left margin), and `smgr` (set right margin).

Delays

Certain capabilities control padding in the `tty`(4) driver. These are primarily needed by hard-copy terminals, and they are used by `tput init` to set tty modes appropriately. You can use delays embedded in the `cr`, `ind`, `cub1`, `ff`, and `tab` capabilities to set the appropriate delay bits to be set in the tty driver. If you specify `pb` (padding baud rate), these values can be ignored at baud rates below the value of `pb`.

Status Lines

If the terminal has an extra ‘‘status line’’ that the software usually does not use, you can indicate this fact. If the status line is viewed as an extra line below the bottom line, into which one can cursor address normally (such as the Heathkit h19’s 25th line, or the 24th line of a VT100, which is set to a 23-line scrolling region), you should specify the `hs` capability. You can specify special strings that go to a given column of the status line and return from the status line as `tsl` and `fsl`. (`fsl` must leave the cursor position in the same place it was in before `tsl`. If necessary, you can include the `sc` and `rc` strings in `tsl` and `fsl` to get this effect.) The `tsl` capability takes one parameter, which is the column number of the status line to which the cursor will be moved.

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

Line Graphics

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

Glyph name	vt100+ character	Glyph name	vt100+ character
Arrow pointing right	+	Upper left corner	l
Arrow pointing left	,	Lower left corner	m
Arrow pointing down	.	Plus	n
Solid square block	0	Scan line 1	o
Lantern symbol	I	Horizontal line	q
Arrow pointing up	-	Scan line 9	s
Diamond	`	Left tee (├)	t
Checker board (stipple)	a	Right tee (┤)	u
Degree symbol	f	Bottom tee (┴)	v
Plus/minus	g	Top tee (┬)	w
Board of squares	h	Vertical line	x
Lower right corner	j	Bullet	~
Upper right corner	k		

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

Example:

Glyph name	vt100+ char	New tty char
Upper left corner	l	R
Lower left corner	m	F
Upper right corner	k	T
Lower right corner	j	G
Horizontal line	q	,
Vertical line	x	.

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

Miscellaneous

If the terminal requires other than a null (0) character as a pad, you can specify this as pad. Only the first character of the pad string is used. If the terminal does not have a pad character, specify npc.

If the terminal can move up or down half a line, you can indicate this with hu (half-line up) and hd (half-line down). This is primarily useful for superscripts and subscripts on hard-copy terminals. If a hard-copy terminal can eject to the next page (form feed), specify this as ff (usually <CONTROL-L>).

If a command to repeat a given character a particular number of times exists (to save time transmitting a large number of identical characters), you can indicate this by using the parameterized string rep. The first parameter is the character to be repeated, and the second is the number of times to repeat it. Thus, tparm(repeat_char, ’x’, 10) is the same as xxxxxxxxxx.

If the terminal has a settable command character, such as the Tektronix 4025, you can indicate this with cmdch. A prototype command character is chosen, which is used in all capabilities. This character is given in the cmdch capability to identify it. The following convention is supported on some UNIX systems: If the CC environment variable exists, all occurrences of the prototype character are replaced with the character in CC.

Terminal descriptions that do not represent a specific kind of known terminal, such as switch, dialup, patch, and network, should include the gn (generic) capability so that programs can complain that they do not know how to talk to the terminal. (This capability does not apply to virtual terminal descriptions for which the escape sequences are known.) If the terminal is one of those supported by the UNIX system virtual terminal protocol, you can specify the terminal number as vt. You should specify a line-turn-around sequence to be transmitted before doing reads in rfi.

If the terminal uses xon/xoff handshaking for flow control, specify xon. You still should include padding information so that routines can make better decisions about costs, but actual pad characters will not be transmitted. You may specify sequences to turn on and off xon/xoff handshaking in smxon and rmxon. If the characters used for handshaking are not ^S and ^Q, you may specify them by using xonc and xoffc.

If the terminal has a ‘‘meta key’’ that acts as a shift key, setting the 8th bit of any character transmitted, you can indicate this fact by using km; otherwise, software assumes that the 8th bit is parity, and it usually is cleared. If strings exist to turn this ‘‘meta mode’’ on and off, you can specify them as smm and rmm.

If the terminal has more lines of memory than will fit on the screen at one time, you can indicate the number of lines of memory by using lm. A value of lm#0 indicates that the number of lines is not fixed, but that still more memory exists than fits on the screen.

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

Special Cases

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

Terminals that cannot display tilde (~) characters, such as certain Hazeltine terminals, should indicate hz.

Terminals that ignore a line feed immediately after an am wrap, such as the Concept 100, should indicate xenl. Those terminals whose cursor remains on the rightmost column until another character has been received, rather than wrapping immediately on receiving the rightmost character, such as the VT100, also should indicate xenl.

If el is required to remove standout (instead of writing normal text on top of it), you should specify xhp.

Those Teleray terminals whose tabs turn all characters moved over to blanks, should indicate xt (destructive tabs). This capability also is taken to mean that it is not possible to position the cursor on top of a ‘‘magic cookie’’; therefore, to erase standout mode, use delete and insert line.

Those Beehive Superbee terminals that do not transmit the escape or <CONTROL-C> characters, should specify xsb, indicating that the <f1> key will be used for escape and the <f2> key for <CONTROL-C>.

Similar Terminals

If two very similar terminals exist, one can be defined as being just like the other with certain exceptions. You can specify the string capability use with the name of the similar terminal. The capabilities given before use override those in the terminal type invoked by use. To cancel a capability, place *xx*@ to the left of the capability definition; *xx* is the capability. For example, the following entry defines an AT&T 4424 terminal that does not have the rev, sgr, and smul capabilities, and hence, it cannot do highlighting. This is useful for different modes for a terminal, or for different user preferences. You may specify more than one use capability.

```
att4424-2|Teletype 4424 in display function group ii,
                        rev@, sgr@, smul@, use=att4424,
```

FILES

`/usr/lib/tabset/*`	Files that contain tab stop settings for some terminals, in a format appropriate to be output to the terminal (escape sequences that set margins and tab stops)
`/usr/lib/terminfo/?/*`	Files that contain terminal descriptions

SEE ALSO

`term`(5)

`tset`(1B) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

`curses`(3) (available only online)

`printf`(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

`tic`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

text_tapeconfig – Tape subsystem configuration file

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The system uses a tape configuration file named text_tapeconfig in the/etc/config directory. If the tpinit(8) command does not find this file, it terminates and returns an error message.

The text_tapeconfig file defines all of the tape devices that the system uses. For detailed information on configuring your devices, see the documentation from the tape device vendors.

The diagnostic devices are implicitly defined when the I/O processors (IOPs) and the channels are defined. You may not redefine them.

The tape configuration file consists of comments (optional) and statements. A comment begins with the # symbol and continues to the end of line. A statement consists of a name followed by a list of keyword parameters. There are four statements; two of these statements also consist of substatements. Statements must be in the order shown:

1. LOADER statements (one per loader)
2. DEVICE_GROUP statements (one per device group)
3. IOP statements (one per IOP) or IONODE statements (one per node)

   The IOP or IONODE statement consists of the following two statements that define the IOP or node configuration:

   a. CHANNEL statements (one per channel in the IOP or node)

   b. BANK statements (one per bank)

      The BANK statement consists of two of the following three statements that define the bank configuration:

      i. SLAVE statements (one per slave device) (IOS-E only)
      or
      CONTROL_UNIT statements (one per control unit)

      ii. DEVICE statements (one per device)
4. OPTIONS statement

Statement Syntax Rules

The following syntax rules apply to text_tapeconfig statements:

- The statement name and its parameters are separated by one or more white spaces (blank, tab, or newline characters).

- Adjacent parameters are separated by a comma.
- The end of the parameter list is indicated by the absence of a comma.
- Adjacent statements are separated by one or more white spaces.

The following is a list of keyword parameter syntax rules:

- The keyword is separated from its value by the = symbol.
- The value of a keyword may consist of keywords, numbers, character strings, and lists of keywords, numbers, and character strings.
- If the value of a keyword is a list, the list is enclosed within left and right parentheses. Adjacent elements of a list are separated by a comma. If the list consists of one element, you do not have to enclose it in parentheses. The elements of a list may be lists.
- Numbers may be specified in decimal, octal, and hexadecimal formats. These formats are the same as those used in the C programming language:

Decimal	First digit is not 0 (1372)
Octal	First digit is 0 (0563)
Hexadecimal	First 2 characters are either 0x or 0X (0xf2)

- Character strings are series of characters. If any one of the special characters (white space, ", #, =, {, }, (,), ′, \ ) is needed in the string, you must enclose the string in a pair of double quotation marks ("). Within a pair of double quotation marks, the sequence of characters will be replaced by *x*; *x* is any character. This is the only way you can specify a " and a \ in a quoted string.
- Comments may appear between any symbols described previously.

You can code the names of statements and keywords in a mixture of uppercase and lowercase letters. The values specified by the user is case sensitive. The following specify the same thing:

```
Name = A
name = A
```

The following are different:

```
name = A
name = a
```

The following are descriptions of the tape configuration statements. You must specify a value for each parameter unless a default is specified or the parameter is described as optional.

LOADER Statement

The LOADER statement identifies the loaders in the text_tapeconfig file and has the following format:

```
LOADER parameter_list
```

A description of the parameters follows:

Parameter	Description
name	Specifies the loader name, which is the object of several tpconfig(8) requests.
type	Specifies the loader type. Currently supported types are as follows:
	EMASS — EMASS autoloader is used.
	IBMTLD — IBM 3494 Tape Library Dataserver is used.
	OPERATOR — Operator loads the drive.
	STKACS — A StorageTek autoloader that is supported by Cray Research is used.
status	Specifies the status (UP or DOWN) of the loader when the tape daemon starts.
message_path_to_loader	Specifies the message path to the servicing loader.
	MSGDAEMON — Uses message daemon to send message to loader.
	NETWORK — Uses TCP/IP protocol to send message to loader.
server	Specifies the server name.
scratch_volume_label_type	Specifies the types of scratch requests that the loader may process. If you specify OPERATOR for the type parameter on the LOADER statement, the following types of scratch requests are available. If you specify any other loader type for the type parameter only NONE is valid.
	AL — ANSI labeled scratch tape requests.
	NL — Nonlabeled scratch tape requests.
	NONE — Scratch labels cannot be used.
	SL — IBM standard labeled scratch requests.
queue_time	Each volume has a designated "best" loader type for the tape mount. If the best loader is not available, this time is used to queue the tape mount request and to wait for the best loader to become available. If the best loader does not become available during this time, the mount request will be issued to the next best loader. A value of 0 indicates to wait up to 24 hours; a nonzero value specifies the number of seconds to wait.
verify_non_label_vsn	Specifies whether the nonlabel VSN should be verified. This parameter may be either YES or NO.
message_route_masks	Routes mount request messages. You can route the mount request message to multiple locations. The list may consist of the following:

FRONTEND Issues the mount message to the front end that may be reached through the connection to TPCNET.

SERVER Issues the mount message to the server station.

UNICOS Issues the mount message to the message daemon. For more information, see msgdaemon(8).

mode Specifies attended mode:

ATTENDED Prompts for operator intervention.

UNATTENDED Assumes negative response for operator intervention.

return_host Specifies the name of the Cray Research host that serves as the return address for the server.

child_program_name *character string*
Specifies the name of the tape daemon loader child program. When the loader is configured UP, the child program is activated. This program must be in the directory from which the tape daemon is activated. The directory is typically /usr/lib/tp.

If child_program_name is omitted, the following values are used:

Loader	Child Program Name
EMASS	esinet
IBMTLD	ibmnet
STKACS	stknet

loader_ring_status
Specifies whether the loader is alerted to ring status.

ALERT Alerts loader to the ring status when a tape is mounted, and checks that the ring status matches the ring status requested by the tape user.

IGNORE Ignores the ring status when a tape is mounted. A logical ring out status is used for a tape that has been requested with a ring out status, but its actual ring status is ring in. The default is ALERT.

network_retry_tries *number*
Specifies the number of times the tape daemon loader child program attempts to send a request over the network to the server after an initial attempt fails. The default for each child program is 5.

network_send_timeout *number*
Specifies the time in seconds during which the tape daemon loader child program tries to send a request over the network to the server. The default for each child program is 3 seconds.

`server_reply_wait_time` *number*

Specifies the time in seconds during which a request that is being processed by the server is kept in a queue by the tape daemon loader child program. If a reply has not been received within this time, the child program queries the state of the outstanding request.

The default value for each child program is 180 seconds. For a StorageTek loader, this value is multplied by the number of Library Storage Modules in the Automated Cartridge System.

You must specify at least one `OPERATOR` type loader in the tape configuration file. If the file does not contain such an entity, the tape daemon in its initialization process creates one assuming the following values:

```
LOADER
name = Operator ,
type = OPERATOR ,
status = UP ,
message_PATH_TO_LOADER = MSGDAEMON ,
message_class = NONE ,
server = "" ,
scratch_volume_label_type = NONE ,
queue_time = 1 ,
verify_non_label_vsn = YES ,
message_route_masks = UNICOS ,
mode = ATTENDED ,
return_host = ""
```

The entry created by the tape daemon shows in the output of the `tpmls`(8) command.

`DEVICE_GROUP` Statement

The tape daemon enforces a resource limit for each user based on the entry for that user in the user database (UDB). The UDB limit applies to resource group numbers, rather than resource or device names. This necessitates a way of mapping the devices configured in the system to the appropriate resource numbers.

The default set of resources (device groups) is determined from the list of devices specified in the `DEVICE_GROUP` statement. The first device group encountered in the list represents resource group number 0. The second device group in the list represents resource group number 1, and so on.

To change this order, specify the `DEVICE_GROUP` statements in the desired order. The first device group name corresponds to resource limit 0. You can list a maximum of eight different device group names.

The mapping specified in this way allows the flexibility of changing the configuration while maintaining a consistent naming convention for device groups that are mapped to the limits set for each user in the UDB.

The `DEVICE_GROUP` statement has the following format:

`DEVICE_GROUP` *parameter_list*

A description of the parameters follows:

Parameter	Description
name	Specifies the device group name.
minlvl	Specifies the minimum mandatory access control (MAC) level for this device group. The default is the system minimum level 0.
maxlvl	Specifies the maximum MAC level for this device group. The default is the system maximum level.
maxcmp	Specifies the maximum MAC compartments for this device group. The default is the system maximum compartments.
avr	Specifies the status (YES or NO) of automatic volume recognition (AVR) for this group. The default is the avr_at_startup option in the OPTIONS statement.
overcommit	Specifies whether (YES or NO) the number of current mount requests can exceed the number of available tape drives. This option overrides the value specified by the overcommit_at_startup parameter in the OPTIONS statement. If you omit this parameter, the default is the value specified by the overcommit_at_startup parameter in the OPTIONS statement. For more information about overcommitted mount requests, see the tpset(8) and tpstat(1) man pages.

IOP Statement

The IOP statement (IOS-E only) specifies the characteristics of an IOP and has the following format:

```
IOP parameter_list { iop_configuration }
```

iop_configuration consists of a series of CHANNEL statements and BANK statements following the keyword parameters. Descriptions of the CHANNEL and BANK statements follow the IOP parameters:

Parameter	Description
number	Specifies the IOP number. For the CRAY J90 series, this parameter is not used and can be set to 0.
cluster	Specifies the cluster number. For the CRAY J90 series, cluster is the IOS number.
type	Specifies the IOP type (IOP_BMX, IOP_ESCON, or IOP_IPI). The CRAY J90 series do not support this parameter.

IONODE Statement

The IONODE statement (GigaRing based systems) specifies the characteristics of a node and has the following format:

```
IONODE parameter_list { ionode_configuration }
```

ionode_configuration consists of a series of CHANNEL statements and BANK statements following the keyword parameters. Descriptions of the CHANNEL and BANK statements follow the IONODE parameters:

Parameter	Description
node	Specifies the node number.
ring	Specifies the ring number.
type	Specifies the node type (IOP_BMX, IOP_ESCON, or IOP_MPN).

CHANNEL Statement (IOP or IONODE Statement)

The CHANNEL statement specifies channel characteristics of an IOP or node and has the following format:

```
CHANNEL parameter_list
```

A description of the parameters follows:

Parameter	Description
address	Specifies the channel address. The following values are valid: IOS-E 30, 32, 34, 36 ELS 22-27, 30-37 BMN 0-1 ESN 0-3 MPN 0-7
microcode_pathname	(IOS-E only) Specifies the path name of the file that contains the channel microcode and must be specified on the first CHANNEL statement of an IOP.
status	Specifies the status (UP or DOWN) of the channel when the tape daemon is started.
adaptor	(IOS-E only) Specifies the channel adapter type (DCA2, FCA2, or TCA2).
timeout	(IOS-E only) Specifies the ER90 time-out value in seconds. If zero is specified, the IOS-E is set to a time-out period of 10 seconds.

BANK Statement (IOP or IONODE Statement)

The BANK statement specifies the bank characteristics of an IOP or node and has the following format:

```
BANK parameter_list { bank_configuration }
```

bank_configuration specifies a series of SLAVE or CONTROL_UNIT statements followed by a series of DEVICE statements. Descriptions of the SLAVE and CONTROL_UNIT statements follow the optional BANK parameter:

Parameter	Description
number	Specifies a bank number that identifies a bank. Valid values are 0 to 63. Default: a bank number will be assigned to a bank.

SLAVE Statement (BANK Statement)

A SLAVE statement (IOS-E only) specifies the characteristics of a slave device and has the following format:

```
SLAVE parameter_list
```

A description of the parameters follows:

Parameter	Description
status	Specifies the status (UP or DOWN) of the slave when the tape daemon is started.
path	Specifies a list of channel address and slave address pairs encoded in parentheses. The parentheses are part of the syntax and must be coded. The channel number is the channel that is connected to the port address in the slave.
reset_timeout	Specifies the reset time-out (in seconds).

CONTROL_UNIT Statement (BANK Statement)

The CONTROL_UNIT statement specifies the characteristics of a control unit and has the following format:

```
CONTROL_UNIT parameter_list
```

A description of the parameters follows:

Parameter	Description	
status	Specifies the status (UP or DOWN) of the control unit when the tape daemon is started.	
protocol	Specifies the protocol for the control unit, as follows:	
	SCSI	Specifies the SCSI protocol.
	ESCON	Specifies the ESCON protocol. This is the only valid protocol for a control unit attached to an ESCON channel.
	INTERLOCK	Specifies interlock protocol.
	STREAMING	Specifies data streaming at 3.0 Mbyte/s.
	STREAMING45	Specifies data streaming at 4.5 Mbyte/s.
path	Specifies a list of channel address and control unit port address pairs. For the CRAY J90 series, this channel number is the address specified in the CHANNEL statement. Specifies a list of channel address and control unit port address pairs. For SCSI (MPN) tape devices, path specifies a channel and SCSIbus pair. The SCSIbus is determined by looking at the appropriate /opt/CYRIion/adm/mic_code file on the system workstation (SWS). For example, if the file contains the following information, path is 3.	

```
sn9132-mpn2, SCSIbus 3 Target 1 Lun [t310], Type = STKSD-3 (VTAPE)
Vendor = STK
Product ID = SD-3
Microcode Rev Level = 0223
Device Min Block Len = 1
Device Max Block Len = 262144
Fixed Block Len = 0(Variable)
Max Block Size = 262144
Default Compression = ON
Ansi Version = 2
Response Format = 2
Attributes = 0x38
```

The channel is restricted to values in the range of 0 through 7 and, by convention, the channel value is the same as the SCSIbus value.

`link_address`
Specifies the link address of the control unit when it is attached by using an ESCON director. It is set to 0 for directly attached (no director) control units.

DEVICE Statement (BANK Statement)

The DEVICE statement specifies the characteristics of a device and has the following format:

DEVICE *parameter_list*

A description of the parameters follows:

Parameter	Description
name	Specifies the device name.
device_group_name	Specifies the name of the device group defined by a DEVICE_GROUP statement.
status	Specifies the initial status (UP or DOWN) of the device.
id	Specifies the hardware device identifier. For SCSI (MPN) tape devices, id specifies a 3-digit octal number, *xyz*. The following values are valid: *x* Is always 0. *y* Specifies the SCSI target; that is, the SCSI ID. *z* Specifies the tape logical unit (lun). This information is in the appropriate SWS /opt/CYRIion/adm/mic_code file. Although fast and wide devices are supported, SCSI IDs are currently limited to values in the range of 0 through 7.
type	Specifies the device type.

For SCSI (MPN) tape devices, the value for `type` is in parenthesis following `Type = type` in the appropriate SWS `/opt/CYRIion/adm/mic_code` file. For example, the `type` value for STKSD-3 is `VTAPE`.

`loader` Specifies the loader name defined in a `LOADER` statement.

`vendor_address`
Specifies the vendor address of the drive in an autoloader.

The format for a StorageTek drive is as follows:

acs#,lsm#,panel#,drive#

The format for an EMASS drive is as follows:

drive#

`facility_address`
Specifies only the ER90 facility address.

`short_timeout`
Specifies the ER90 short time-out (in seconds).

`long_timeout`
Specifies the ER90 long time-out (in seconds).

`timeout` Specifies the time-out value in seconds that the ESCON IOP waits for a response from the channel. An integer from 1 to 65535 specifies the number of seconds. A value of 0 directs the tape subsystem to use the time-out value that is hard-coded in the ESCON IOP software. This value is currently set to 900 seconds (15 minutes).

`OPTIONS` Statement

The options in force when the tape daemon is built are specified in the `/usr/include/tapedef.h` file. You can specify most of these options in the `OPTIONS` section of the `text_tapeconfig` file.

To override the value with which the tape daemon was built, specify the following options and their corresponding values. Descriptions of the options in the `/usr/include/tapedef.h` file are given in the *Tape Subsystem Administration*, Cray Research publication SG–2307. The options that you can specify in the `text_tapeconfig` file with the `OPTIONS` statement are similar to the options in `tapedef.h`, but not identical. Values are often given in a different form in the two files (for example, the value for the `ask_blp` keyword is expressed as 0 or 1 in `tapedef.h`, but it is expressed as `YES` or `NO` in `text_tapeconfig`).

The format of the OPTIONS statement follows:

OPTIONS *parameter_list*

The following parameter list includes valid values or brief definitions of the options.

Parameter	Description
allow_unprotected	Allows access (YES or NO) to tapes that do not contain a MAC label in the header. Default: YES
ask_label_switch	Seeks permission (YES or NO) from the operator to switch label type. Default: YES
ask_vsn	Seeks permission (YES or NO) from the operator to specify a VSN when a nonlabel tape is mounted. Default: YES
avr_at_startup	Starts (YES or NO) AVR when the tape daemon is started. Default: YES
blocksize	Specifies the block size to use when the user does not specify a block size by using the tpmnt(1) command -b option. Default: 32768
blp_ring_status	Specifies the user status for the use of the -r option of the tpmnt(1) command when the user requests bypass label processing. UNRESTRICTED specifies the user can use both -r in and -r out. OUT specifies the user can use only -r out. Default: UNRESTRICTED.
check_expiration_date	Specifies whether the operator should check and confirm (YES or NO) the expiration date on the header label of a labeled tape. Default: YES
check_file_id	Specifies whether the file ID on a labeled tape should be checked (YES or NO) when the file is opened. Default: YES
check_protection	Specifies whether the protection flag on the header should be checked (YES or NO). Default: YES
check_vsn	Specifies whether the VSN on labeled tapes should be checked (YES or NO). Default: YES
cray_reel_librarian	Specifies whether the Cray/REELlibrarian system is enabled (YES or NO). Default: NO
cray_reel_librarian_mandatory	Specifies whether the Cray/REELlibrarian system is mandatory (YES or NO). Default: NO
cray_reel_librarian_operator_select_scratch	Indicates whether the operator should verify (YES or NO) the scratch mounts by the Cray/REELlibrarian system before continuing. Default: NO

`cray_reel_librarian_scratch_vsn`
Specifies the scratch VSN that the Cray/REELlibrarian system will use to tell the operator that a scratch volume is needed. Default: `?CRL??`

`device_group_name`
Specifies the default device group name if it is not specified on the `-g` option of the `tpmnt`(1) command. Default: `CART`

`file_status` Specifies the file status (`NEW` or `OLD`) if it is not specified on the `tpmnt`(1) command. Default: `OLD`

`label_type` Specifies the label type (`AL`, `SL`, or `NL`) if it is not specified on the `tpmnt`(1) command. Default: `AL`

`loader_device_assignment_order`
Specifies the method (`DEVICE_LIST` or `ROUND_ROBIN`) with which the loader assigns devices. Default: `ROUND_ROBIN`

`mainframe_job_origin`
Specifies the mainframe ID of the job if it is not specified. Default: `C1`

`max_blocksize` Sets the upper limit of the block size of the `-b` parameter on the `tpmnt`(1) command. If you specify a larger value, the command terminates abnormally. Default: 4194303

`max_number_of_device_groups`
Specifies the maximum number of device groups. Default: 8

`max_number_of_tape_users`
Specifies the maximum number of tape users. Default: 64

`message_daemon_pipename`
Specifies the message daemon pipe name. Default: `/usr/spool/msg/msg.requests`

`number_of_autoloader_retries`
Specifies the number of times to try to send a request to the autoloader before informing the operator of an error. The CRAY J90 series do not support this parameter. Default: 10

`operator_message_destination`
Specifies where operator messages are sent; `UNICOS`, `SERVER`, and `FRONTEND`. Default: `(UNICOS)`

`operator_message_frontend_id`
Specifies the front-end ID for operator messages. Default: `" "`

`overcommit_at_startup`
Specifies whether overcommitted mount requests should be enabled as part of start-up when the tape daemon is started (`YES` or `NO`). This option applies only if you omit the `overcommit` parameter on the `DEVICE_GROUP` statement. Default: `NO`

`overcommit_max` Specifies the maximum number of overcommitted mount requests that the tape subsystem can issue. When the number of tape mount requests exceeds this number, the system stops processing requests until one or more of the already overcommitted mount requests are satisfied. You may change this setting by using the `tpset`(8) command. Default: 20

`reselect_cart` Specifies whether another device will be selected (`YES` or `NO`) at end-of-volume for cartridge type devices, which include 3480, 3490, and 3490E devices. Default: `NO`

`retention_period_days`
Specifies the retention period (in days). Default: 0

`ring_status` Specifies the ring status when the ring option (`-r`) is not specified on the `tpmnt`(1) command (`IN`, `OUT`, or `(IN,OUT)`). Default: `(IN,OUT)`

`scratch_volume_action`
Specifies the action (`FREE` or `KEEP`) to perform for scratch tapes when they are released. Default: `FREE`

`scratch_volume_retries`
Specifies the number of retries to get a scratch volume out of the autoloader scratch pool. Default: 3

`scratch_volume_vsn`
Specifies the scratch tape VSN. Default: `??????`

`secure_frontend` Specifies whether security on the front end is enabled (`YES` or `NO`). The CRAY J90 series do not support front-end servicing. Default: `" "`
`servicing_frontend_at_startup` Specifies whether front-end servicing should start (`YES` or `NO`) when the tape daemon is started. The CRAY J90 series do not support any front-end servicing. Default: `NO`

`servicing_frontend_id`
Specifies the servicing front-end ID to use when the `-m` option is missing on the `tpmnt`(1) command. The CRAY J90 series do not support front-end servicing. Default: `" "`

`servicing_frontend_mandatory`
Specifies whether the front-end ID specified by the `servicing_frontend_id` parameter is used (`YES` or `NO`) regardless of the `-m` option on the `tpmnt`(1) command. The CRAY J90 series do not support front-end servicing. Default: `NO`

`servicing_frontend_protocol`
Specifies the protocol (`TCP`) to talk to front ends. The CRAY J90 series do not support front-end servicing. Default: `TCP`

`system_code` Specifies the system code to put on tape labels. Default: `CRI/UNICOS`

`tcp_daemon_childname`
Specifies the child name of the TCP daemon. The CRAY J90 series do not support this parameter. Default: `tcpnet`

`tcp_daemon_frontend_id`
Specifies the front-end ID of the TCP daemon. Default: `" "`

`tape_daemon_trace_file_group_id`
Specifies the group ID of the tape daemon trace files. Default: 9

`tape_daemon_trace_file_mode`
Specifies the file mode of the tape daemon trace files. Default: 0640

`tape_daemon_trace_file_owner`
Specifies the owner ID of the tape daemon trace files. Default: 0

`tape_daemon_trace_file_prefix`
Specifies the tape daemon trace file prefix. Default: `/usr/spool/tape/trace`

`tape_daemon_trace_file_size`
Specifies the size (in bytes) of the tape daemon trace files. Default: 409600

`tape_daemon_trace_flg`
Specifies whether tape tracing is enabled (`YES` or `NO`). Default: `YES`

`tape_daemon_trace_savefile_prefix`
Specifies the prefix to the tape daemon save files. Default: `/usr/spool/tape/trace`

`tcp_daemon_pipename`
Specifies the pipe name of the TCP daemon. The CRAY J90 series do not support this parameter. Default: `/usr/spool/tape/tcpnet.pipe`

`tcp_daemon_socket_port_number`
Specifies the socket port number of the TCP daemon. Default: 1167

`user_exit_mask` Enables the use of the listed user exits. If no user exits are required, this entry is not needed. For a list of user exits, see the *Tape Subsystem Administration*, Cray Research publication SG–2307. Default: `UEX_NONE`

`verify_scratch_vsn`
Indicates (`YES` or `NO`) that you may need to send the operator a message that requests verification that a scratch tape is being used to satisfy a tape mount request. You must consult the operator if front-end servicing is not in use. Default: `YES`

EXAMPLES

The following two examples show `text_tapeconfig` files used on IOS-E systems and GigaRing based systems.

For more detail, check the documentation from your tape device vendors; configuration possibilities vary depending up the vendors and the specific devices that you are configuring. For example, an IBM 3490E controller supports multiple devices while a StorageTek Redwood only supports a single device.

GigaRing based systems

The following `text_tapeconfig` file illustrates some typical configurations for GigaRing based systems.

```
LOADER
        name = Operator ,
        type = OPERATOR ,
        status = UP ,
        message_path_to_loader = MSGDAEMON ,
        message_class = NONE ,
        server = UNICOS ,
        scratch_volume_label_type = (NL,AL,SL) ,
        queue_time = 0 ,
        verify_non_label_vsn = NO ,
        message_route_masks = (UNICOS) ,
      loader_ring_status = ALERT,
        mode = ATTENDED
LOADER
        name = stksun ,
        type = STKACS ,
        status = DOWN ,
        message_path_to_loader = NETWORK ,
        message_class = TYPE_340 ,
        server = robot ,
        scratch_volume_label_type = NONE ,
        queue_time = 180 ,
        verify_non_label_vsn = NO ,
        message_route_masks = (UNICOS, FRONTEND) ,
      loader_ring_status = IGNORE,
        mode = ATTENDED

LOADER
        name = wolfy ,
        type = STKACS ,
        status = DOWN ,
        mode = ATTENDED ,
        message_class = TYPE_340 ,
        message_path_to_loader = NETWORK ,
        server = 9490ldr ,
        scratch_volume_label_type = NONE ,
        queue_time = 0 ,
        verify_non_label_vsn = NO ,
```

```
      loader_ring_status = IGNORE,
        message_route_masks = (FRONTEND,UNICOS)

LOADER
        name = panther ,
        type = STKACS ,
        status = DOWN ,
        mode = ATTENDED ,
        message_class = TYPE_340 ,
        message_path_to_loader = NETWORK ,
        server = stk9710 ,
        scratch_volume_label_type = NONE ,
        queue_time = 0 ,
        verify_non_label_vsn = NO ,
      loader_ring_status = IGNORE,
        message_route_masks = (FRONTEND,UNICOS)

LOADER
        name = ibm ,
        type = IBMTLD ,
        status = DOWN ,
        message_path_to_loader = NETWORK ,
        message_class = TYPE_340 ,
        server = ibmtld ,
        scratch_volume_label_type = NONE ,
        queue_time = 15 ,
        verify_non_label_vsn = NO ,
        message_route_masks = UNICOS ,
      loader_ring_status = IGNORE,
        mode = ATTENDED

DEVICE_GROUP
      name = DEFAULT ,
      avr = YES,
      overcommit = NO
DEVICE_GROUP
      name = DAT ,
      avr = YES ,
      overcommit = NO
DEVICE_GROUP
        name = IBM3590 ,
        avr = YES,
      overcommit = NO
DEVICE_GROUP
```

```
        name = IBM3490E ,
        avr = YES ,
      overcommit = NO
DEVICE_GROUP
        name = STK4890 ,
        avr = YES,
      overcommit = NO
DEVICE_GROUP
      name = DLT4000 ,
      avr = YES,
      overcommit = NO
DEVICE_GROUP
        name = STK9490 ,
        avr = YES,
      overcommit = NO
DEVICE_GROUP
      name = STKSD3 ,
      avr = YES,
      overcommit = NO
IONODE
      node     = 1 ,
      ring     = 0 ,
      type     = IOP_MPN
      {
            CHANNEL
                  address = 0 ,
                  status  = up
            CHANNEL
                  address = 1 ,
                  status  = up
            CHANNEL
                  address = 2 ,
                  status  = up
            CHANNEL
                  address = 3 ,
                  status  = up
            CHANNEL
                  address = 4 ,
                  status  = up
            CHANNEL
                  address = 5 ,
                  status  = up
            CHANNEL
                  address = 6 ,
```

```
      status  = up
CHANNEL
      address = 7 ,
      status  = up
    BANK
           number = 1
           {
                  CONTROL_UNIT
                         status = UP ,
                         path = (1,1) ,
                         protocol = SCSI
                  DEVICE
                         name = s4890s0,
                 device_group_name = STK4890,
                         id = 000 ,
                         type = 3490E ,
                         status = DOWN ,
                 vendor_address = (0,0,2,0) ,
                         loader = panther
                  DEVICE
                         name = s4890s1,
                   device_group_name = STK4890,
                         id = 010 ,
                         type = 3490E ,
                         status = DOWN ,
                      vendor_address = (0,0,2,1) ,
                         loader = panther
                  DEVICE
                         name = d4000s0,
                  device_group_name = DLT4000,
                         id = 020 ,
                         type = VTAPE ,
                         status = DOWN ,
                  vendor_address = (0,0,2,2) ,
                           loader = panther
                    DEVICE
                           name = d4000s1,
                    device_group_name = DLT4000,
                           id = 030 ,
                           type = VTAPE ,
                           status = DOWN ,
                  vendor_address = (0,0,2,3) ,
                           loader = panther
           }
```

```
                BANK
                        number = 6
                        {
                                CONTROL_UNIT
                                        status = UP ,
                                        path = (6,6),
                                        protocol = SCSI
                                DEVICE
                                        name =  3490s0,
                                        device_group_name = IBM3490E ,
                                        id = 060,
                                        type = 3490E,
                                        status = DOWN ,
                                        loader = ibm
                                DEVICE
                                        name =  3490s1,
                                        device_group_name = IBM3490E ,
                                        id = 061,
                                        type = 3490E,
                                        status = DOWN ,
                                        loader = ibm
                        }
        }
IONODE
        node     = 1 ,
        ring     = 1 ,
        type     = IOP_MPN
        {
                CHANNEL
                        address = 0 ,
                        status  = up
                CHANNEL
                        address = 1 ,
                        status  = up
                CHANNEL
                        address = 2 ,
                        status  = up
                CHANNEL
                        address = 3 ,
                        status  = up
                CHANNEL
                        address = 4 ,
                        status  = up
                CHANNEL
```

```
      address = 5 ,
      status  = up
CHANNEL
      address = 6 ,
      status  = up
CHANNEL
      address = 7 ,
      status  = up
BANK
      number = 17
      {
            CONTROL_UNIT
                  status    = up ,
                  path      = (7,7) ,
                  protocol  = SCSI
                    DEVICE
                            name =  d5649JX,
                            device_group_name = DAT ,
                            id = 040 ,
                            type = VTAPE,
                            status = DOWN ,
                            loader = Operator
      }
BANK
      number = 16
      {
            CONTROL_UNIT
                  status    = up ,
                  path      = (6,6) ,
                  protocol  = SCSI
                    DEVICE
                            name =  s9490s0,
                            device_group_name = STK9490 ,
                            id = 000 ,
                            type = 3490E,
                            status = down ,
                            vendor_address = (0,0,1,0) ,
                            loader = wolfy
                    DEVICE
                            name =  s9490s1,
                            device_group_name = STK9490 ,
                            id = 010 ,
                            type = 3490E,
                            status = DOWN ,
```

```
                                vendor_address = (0,0,1,1) ,
                                loader = wolfy
                        DEVICE
                                name =  s9490s2,
                                device_group_name = STK9490 ,
                                id = 020 ,
                                type = 3490E,
                                status = DOWN ,
                                vendor_address = (0,0,1,2) ,
                                loader = wolfy
                        DEVICE
                                name =  s9490s3,
                                device_group_name = STK9490 ,
                                id = 030 ,
                                type = 3490E,
                                status = DOWN ,
                                vendor_address = (0,0,1,3) ,
                                loader = wolfy
        }
        BANK
                number = 10
                {
                        CONTROL_UNIT
                                status = UP ,
                                path = (0, 0),
                                protocol = SCSI
                        DEVICE
                                name =  3590s0,
                                device_group_name = IBM3590 ,
                                id = 000,
                                type = VTAPE,
                                status = DOWN ,
                                loader = ibm
        }
        BANK
                number = 12
                {
                        CONTROL_UNIT
                                status = UP ,
                                path = (2, 2),
                                protocol = SCSI
                        DEVICE
                                name =  3590s1,
                                device_group_name = IBM3590 ,
```

```
                                        id = 010,
                                        type = VTAPE,
                                        status = DOWN ,
                                        loader = ibm
                    }
                    BANK
                          number = 11
                          {
                           CONTROL_UNIT
                                 status    = up ,
                                 path      = (1,1) ,
                                 protocol  = SCSI
                                  DEVICE
                                          name =  ssd3_s0,
                                          device_group_name = STKSD3 ,
                                          id = 010 ,
                                          type = VTAPE,
                                          status = DOWN ,
                                 vendor_address = (0,0,3,1) ,
                                         loader = wolfy
                    }
                    BANK
                          number = 13
                          {
                           CONTROL_UNIT
                                 status    = up ,
                                 path      = (3,3) ,
                                 protocol  = SCSI
                                  DEVICE
                                          name =  ssd3_s1,
                                          device_group_name = STKSD3 ,
                                          id = 030 ,
                                          type = VTAPE,
                                          status = DOWN ,
                                 vendor_address = (0,0,3,3) ,
                                          loader = wolfy
                    }
            }

      OPTIONS
      allow_unprotected          = YES ,
      ask_label_switch           = YES ,
      ask_vsn                         = NO ,
      avr_at_startup                  = YES ,
```

```
blp_ring_status                    = UNRESTRICTED ,
blocksize                  = 65536 ,
check_expiration_date              = YES ,
check_file_id                      = YES ,
check_protection           = NO ,
check_vsn                  = YES ,
cray_reel_librarian                = NO ,
cray_reel_librarian_mandatory      = NO ,
cray_reel_librarian_operator_select_scratch    = NO ,
cray_reel_librarian_scratch_vsn          = ?CRL?? ,
device_group_name          = DEFAULT ,
file_status                = OLD ,
label_type                 = AL ,
loader_device_assignment_order           = ROUND_ROBIN ,
mainframe_job_origin               = C1 ,
max_number_of_device_groups        = 8 ,
max_blocksize                      = 4194303 ,
max_number_of_tape_users           = 100 ,
message_daemon_pipename            = /usr/spool/msg/msg.requests ,
number_of_autoloader_retries       = 10 ,
operator_message_destination       = UNICOS ,
operator_message_frontend_id       = "" ,
operator_message_type              = USCP_TYPE_1 ,
overcommit_at_startup              = NO,
overcommit_max                     = 20,
reselect_cart                      = NO ,
retention_period_days              = 0 ,
ring_status                = (IN,OUT) ,
scratch_volume_action              = FREE ,
scratch_volume_retries             = 3 ,
scratch_volume_vsn                 = ?????? ,
secure_frontend            = NO ,
servicing_frontend_at_startup      = NO ,
servicing_frontend_id              = "" ,
servicing_frontend_mandatory       = NO ,
servicing_frontend_protocol        = TCP ,
stop_hippi_eiop                    = YES ,
system_code                        = CRI/UNICOS ,
tape_daemon_trace_file_group_id    = 9 ,
tape_daemon_trace_file_mode        = 0640 ,
tape_daemon_trace_file_owner       = 0 ,
tape_daemon_trace_file_prefix      = /usr/spool/tape/trace ,
tape_daemon_trace_file_size        = 409600 ,
tape_daemon_trace_flg                 = YES ,
```

```
  tape_daemon_trace_savefile_prefix   = /usr/spool/tape/save/trace ,
  tcp_daemon_childname                = tcpnet ,
  tcp_daemon_frontend_id              = "mvs" ,
  tcp_daemon_pipename                 = /usr/spool/tape/tcpnet.pipe ,
  tcp_daemon_socket_port_number       = 1167 ,
  user_exit_mask                      = (UEX_NONE) ,
  verify_scratch_vsn                  = YES
```

IOS-E support

The following `text_tapeconfig` file illustrates some typical configurations for systems with IOS-E support.

```
LOADER
        name = Operator ,
        type = OPERATOR ,
        status = UP ,
        mode = ATTENDED ,
        message_path_to_loader = MSGDAEMON ,
        server = UNICOS ,
        scratch_volume_label_type = (NL,AL,SL) ,
        queue_time = 0 ,
        verify_non_label_vsn = YES ,
        message_route_masks = (UNICOS,FRONTEND)

LOADER
        name = stksun ,
        type = STKACS ,
        status = DOWN ,
        mode = ATTENDED ,
        message_path_to_loader = NETWORK ,
        server = robot ,
        scratch_volume_label_type = NONE ,
        queue_time = 15 ,
        verify_non_label_vsn = NO ,
        message_route_masks = (UNICOS)

LOADER
        name = esisun ,
        type = EMASS ,
        status = DOWN ,
        mode = ATTENDED ,
        message_path_to_loader = NETWORK ,
        server = er90-sun ,
        scratch_volume_label_type = NONE ,
        queue_time = 15 ,
```

```
        verify_non_label_vsn = NO ,
        message_route_masks = (UNICOS)

DEVICE_GROUP
        name = CART

DEVICE_GROUP
        name = TAPE

DEVICE_GROUP
        name = SILO

DEVICE_GROUP
        name = 3490

DEVICE_GROUP
        name = 3490E ,
        avr  = YES

DEVICE_GROUP
        name = TEST

DEVICE_GROUP
        name = ER90

DEVICE_GROUP
        name = ESCON

IOP
        number   = 3 ,
        cluster  = 0 ,
        type     = IOP_IPI
        {
                CHANNEL
                        address = 036 ,
                        status  = UP ,
                        microcode_pathname = /etc/micro/IPI3.ucode ,
                        timeout = 10000 ,
                        adaptor = DCA2
                CHANNEL
                        address = 034 ,
                        status  = UP ,
                        microcode_pathname = /etc/micro/IPI3.ucode ,
                        timeout = 10000 ,
```

```
                        adaptor = DCA2
                BANK
                        number = 1
                        {
                                SLAVE
                                        path = (034,0) ,
                                        status = UP ,
                                        reset_timeout = 1000
                                DEVICE
                                        name   = er92 ,
                                        device_group_name = ER90 ,
                                        id     = 0 ,
                                        type   = ER90 ,
                                        status = DOWN ,
                                        loader = esisun ,
                                        vendor_address = 2 ,
                                        facility_address = 0xFF ,
                                        short_timeout = 600 ,
                                        long_timeout  = 400
                        }
                BANK
                        number = 2
                        {
                                SLAVE
                                        path = (036,0) ,
                                        status = UP ,
                                        reset_timeout = 1000
                                DEVICE
                                        name   = er93 ,
                                        device_group_name = ER90 ,
                                        id     = 0 ,
                                        type   = ER90 ,
                                        status = DOWN ,
                                        loader = esisun ,
                                        vendor_address = 3 ,
                                        facility_address = 0xFF ,
                                        short_timeout = 600 ,
                                        long_timeout  = 400
                        }
        }
IOP
        number   = 1 ,
        cluster  = 0 ,
        type     = IOP_BMX
```

```
{
        CHANNEL
                address = 030 ,
                status  = UP ,
                microcode_pathname = /etc/micro/TCA1.ucode
        CHANNEL
                address = 036 ,
                status  = UP ,
                microcode_pathname = /etc/micro/TCA1.ucode
        CHANNEL
                address = 034 ,
                status  = UP ,
                microcode_pathname = /etc/micro/TCA1.ucode
        CHANNEL
                address = 032 ,
                status  = UP ,
                microcode_pathname = /etc/micro/TCA1.ucode
        BANK
                number = 3
                {
                        CONTROL_UNIT
                                status    = UP ,
                                path      = ((036, 0)) ,
                                protocol  = INTERLOCK
                        DEVICE
                                name   = 220 ,
                                device_group_name = TAPE ,
                                id     = 00 ,
                                type   = 3420 ,
                                status = DOWN ,
                                loader = Operator
                        DEVICE
                                name   = 221 ,
                                device_group_name = TAPE ,
                                id     = 01 ,
                                type   = 3420 ,
                                status = DOWN ,
                                loader = Operator
                }

        BANK
            number = 4
                {
                        CONTROL_UNIT
```

```
                                        status    = UP ,
                                        path      = ((034,11),(030,11),(032,0)) ,
                                        protocol  = STREAMING45
                                DEVICE
                                        name   = 120 ,
                                        device_group_name = CART ,
                                        id     = 00 ,
                                        type   = 3480 ,
                                        status = DOWN ,
                                        loader = Operator
                                DEVICE
                                        name   = 300 ,
                                        device_group_name = CART ,
                                        id     = 00 ,
                                        type   = 3480 ,
                                        status = DOWN ,
                                        loader = stksun ,
                                        vendor_address = (0,0,10,0)
                                DEVICE
                                        name   = 301 ,
                                        device_group_name = CART ,
                                        id     = 01 ,
                                        type   = 3480 ,
                                        status = DOWN ,
                                        loader = stksun ,
                                        vendor_address = (0,0,10,1)
                                DEVICE
                                        name   = 170 ,
                                        device_group_name = 3490E ,
                                        id     = 04 ,
                                        type   = 3490E ,
                                        status = DOWN ,
                                        loader = Operator
                }
        }
IOP
        number   = 3 ,
        cluster  = 1 ,
        type     = IOP_ESCON
        {
                CHANNEL
                        address = 036 ,
                        status  = UP ,
                        microcode_pathname = /etc/micro/FCA2.ucode ,
```

```
                adaptor = FCA2
        CHANNEL
                address = 034,
                status  = UP ,
                microcode_pathname = /etc/micro/FCA2.ucode,
                adaptor = FCA2
         BANK
                number = 5
{
                        CONTROL_UNIT
                                status    = UP ,
                                path      = ((036, 00)) ,
                                protocol  = ESCON ,
                                link_address = 0

                        DEVICE
                                name   = 170e ,
                                device_group_name = ESCON ,
                                id     = 00 ,
                                type   = 3490E ,
                                status = DOWN ,
                                loader = operator
}

        BANK
                number = 6
{
                        CONTROL_UNIT
                                status    = UP ,
                                path      = ((034, 00)) ,
                                protocol  = ESCON ,
                                link_address = C3

                        DEVICE
                                name   = 34C300 ,
                                device_group_name = ESCON ,
                                id     = 00 ,
                                type   = 3490E ,
                                status = DOWN ,
                                loader = operator

                        DEVICE
                                name   = 34C301 ,
                                device_group_name = ESCON ,
```

```
                                        id     = 01 ,
                                        type   = 3490E ,
                                        status = DOWN ,
                                        loader = operator
        }
                BANK
                        number = 7
        {
                                CONTROL_UNIT
                                        status    = UP ,
                                        path = ((034, 00)),
                                        protocol = ESCON ,
                                        link_address = C8

                                DEVICE
                                        name   = 34C800 ,
                                        device_group_name = ESCON ,
                                        id     = 00 ,
                                        type   = 3490E ,
                                        status = DOWN ,
                                        loader = operator
        }
                BANK
                        number = 8
        {
                                CONTROL_UNIT
                                        status    = UP ,
                                        path      = ((030, 00)),
                                        protocol  = ESCON ,
                                        link_address = C3

                                CONTROL_UNIT
                                        path      = ((032, 00)) ,
                                        status    = UP ,
                                        protocol  = ESCON ,
                                        link_address = C3

                                DEVICE
                                        name   = device1 ,
                                        device_group_name = ESCON ,
                                        id     = 00 ,
                                        type   = 3490E ,
                                        status = DOWN ,
                                        loader = operator
```

```
        }
OPTIONS
allow_unprotected                       = YES,
ask_label_switch                        = YES ,
ask_vsn                                 = YES ,
avr_at_startup                          = YES ,
blp_ring_status                         = UNRESTRICTED ,
blocksize                               = 32768 ,
check_expiration_date                   = YES ,
check_file_id                           = YES ,
check_protection                        = YES ,
check_vsn                               = YES ,
cray_reel_librarian                     = NO ,
cray_reel_librarian_mandatory           = NO ,
cray_reel_librarian_operator_select_scratch     = NO ,
cray_reel_librarian_scratch_vsn         = ?CRL?? ,
device_group_name                       = CART ,
file_status                             = OLD ,
label_type                              = AL ,
loader_device_assignment_order          = ROUND_ROBIN ,
mainframe_job_origin                    = C1 ,
max_number_of_device_groups             = 8 ,
max_blocksize                           = 4194303 ,
max_number_of_tape_users                = 64 ,
message_daemon_pipename                 = /usr/spool/msg/msg.requests ,
number_of_autoloader_retries            = 10 ,
operator_message_destination            = (UNICOS) ,
operator_message_frontend_id            = "" ,
overcommit_at_startup                   = NO ,
overcommit_max                          = 20 ,
reselect_cart                           = NO ,
retention_period_days                   = 0 ,
ring_status                             = (IN,OUT) ,
scratch_volume_action                   = FREE ,
scratch_volume_retries                  = 3 ,
scratch_volume_vsn                      = ?????? ,
secure_frontend                         = NO ,
servicing_frontend_at_startup           = NO ,
servicing_frontend_id                   = "" ,
servicing_frontend_mandatory            = NO ,
servicing_frontend_protocol             = TCP ,
system_code                             = CRI/UNICOS ,
tape_daemon_trace_file_group_id         = 9 ,
tape_daemon_trace_file_mode             = 0640 ,
```

```
tape_daemon_trace_file_owner            = 0 ,
tape_daemon_trace_file_prefix           = /usr/spool/tape/trace ,
tape_daemon_trace_file_size             = 409600 ,
tape_daemon_trace_flg                   = YES ,
tape_daemon_trace_savefile_prefix       = /usr/spool/tape/save/trace ,
tcp_daemon_childname                    = tcpnet ,
tcp_daemon_frontend_id                  = " " ,
tcp_daemon_pipename                     = /usr/spool/tape/tcpnet.pipe ,
tcp_daemon_socket_port_number           = 1167 ,
user_exit_mask                          = UEX_NONE ,
verify_scratch_vsn                      = YES ,
```

FILES

`/etc/config/text_tapeconfig`	Tape subsystem configuration file
`/usr/include/tapedef.h`	Definitions for trace file size
`/usr/include/tapereq.h`	Tape daemon interface definition file

SEE ALSO

`msgdaemon`(8), `tpconf`(8), `tpconfig`(8), `tpinit`(8), `tpmls`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

Tape Subsystem Administration, Cray Research publication SG–2307

NAME

tmpdir.users – List of authorized users for tmpdir(1)

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The /etc/tmpdir.users file contains a list of users and their authorizations for the tmpdir(1) command.

The tmpdir.users file is an ASCII file that contains one entry for each authorized user. Entries have the following format:

user_name : *path_name* [: *path_name*]

The first field in each line is the user name (login name). The remaining fields specify the path names of directories in which the user may create temporary directories. The fields are separated by colons; each entry is separated from the next by a new-line character.

FILES

/etc/tmpdir.users List of authorized users for tmpdir(1)

SEE ALSO

tmpdir(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

NAME

types – Definition of primitive system data types

SYNOPSIS

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

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

The data types used in UNICOS system code are defined in the sys/types.h include file. Some data of these types is accessible to user code.

```
typedef long           word;
typedef unsigned long  ulong;
typedef unsigned int   uint;
typedef unsigned short ushort;
typedef long           blkno_t;
typedef long           daddr_t;
typedef struct inode   inode_t;
typedef word           label_t[128];  /* save area for Bs,Ts */

typedef word *         waddr_t;
typedef unsigned char  uchar;
typedef short          cnt_t;
typedef long           paddr_t;
typedef long           key_t;
```

The daddr_t format is used for disk addresses except in an inode on disk; see fs(5). Times are encoded in seconds since 00:00:00 GMT, January 1, 1970. The major and minor parts of a device code specify the kind and unit number of a device and are installation-dependent. Offsets are measured in bytes from the beginning of a file. The label_t array of variables is used to save the processor state while another process is running.

FILES

/usr/include/sys/types.h Data types definition file

SEE ALSO

fs(5)

NAME

udb – Format of the user database (UDB) file

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The user database (UDB) files contain control information for users of the UNICOS operating system and for the fair-share scheduler's resource groups. The UDB files replace the /etc/passwd file as the primary source for user validation and control information. The UDB consists of the following files:

- /etc/udb
- /etc/udb.public
- /etc/udb_2/udb.index
- /etc/udb_2/udb.priva
- /etc/udb_2/udb.pubva

The files in the /etc/udb_2 directory extend the capability of the UDB beyond what was available in previous releases.

To allow users access to nonsensitive UDB information, the /etc/udb.public, /etc/udb_2/index, and /etc/udb_2/udb.pubva files are publicly readable. The other files contain privileged information, such as encrypted passwords and security information, and only privileged callers can read them. Write access to all files is restricted to privileged users.

You should write these files only through the supplied library routines described in libudb(3C). Because of the reliance on file locking to maintain the integrity of the files, other access methods may corrupt the database and require regeneration of the files.

Organization of /etc/udb

The /etc/udb file is organized in blocks of 4096 characters (one sector) to provide the ability to ensure atomic updates of information. The file is organized in blocks, as follows:

Block	Description
0	File header for validation, version control, and default information
1-4	User ID (UID) hash blocks
5-8	Name hash blocks
9-*n*	User information blocks

The user information blocks contain a bidirectional chain that links records in numeric order by UID. This provides the mechanism that implements the next UID and previous UID access functions.

When new records are added to the file, the first empty record slot is allocated and the new record is written to that position. Then the linkage is adjusted to place the new record in the correct logical place in the user ID space. If no empty slot exists, the file will be extended by one block and the new record added at the end of the file. The released library configuration specifies three user records per block.

Organization of extension files

The extension UDB files exist in the `/etc/udb_2` directory. The three extension files are `udb.index`, `udb.priva`, and `udb.pubva`. The format of these files is described in this subsection, following a description of the common file header. You can find the formal declarations of these files in the file `libc/udb/libudb.h`.

Common file header

Each file has a 4096-byte header that contains control information used by the access method. Important information in the header data includes the magic number, which identifies the file, and the version identifier, which identifies the structure of the file. The header contains space for the default UDB table (`struct udbdefault`) and the tape name table (`struct udbtmap`), but this space is used only in `udb.pubva`.

Index file `udb.index`

The `udb.index` index file consists of a common file header, an index header, and two index arrays. The first index array is the name array, and the second is the UID array. The size of the index file depends on the number of records in the database.

The index arrays are packed together following the index header; the entire file occupies one or more 4096-byte blocks with possible free space at the end. The `length` field in the index header reflects this length. The `entries` field in the index header specifies the number of entries actually in use in both arrays. (Each array contains the same number of entries.)

The name array is sorted in ascending order on the `name` field, as determined by `strcmp`(3C); duplicate entries are not allowed. The UID array is sorted on ascending value of UID; duplicates are allowed but the order is arbitrary.

In each index entry, the `pub_pos` field specifies the disk offset of the target record in the public file; the `priv_pos` field is the offset of the start of the record in the private file.

Data files `udb.priva` and `udb.pubva`

The `udb.priva` and `udb.pubva` data files have the same format. The `udb.pubva` file contains public information, and `udb.priva` contains private information. A data file begins with a common file header, followed by an arbitrary number of free records and data records.

Free records are chained together and linked to the free-chain pointer in the common data header. Free records are used, if possible, when new records are created or when a record expands and must be moved to find sufficient contiguous space.

A data record begins with a header that contains the following information:

- Name and UID of the record
- Length and compression information
- Time the record was last changed

- Other control information
- Compressed data fields

The header is designed to provide sufficient information to reconstruct a damaged database without having to decompress the data.

The compressed data is a variable-sized extension to the record header that has been compressed into a bit stream to reduce its size and to remove unnecessary fields. All zero-length fields are deleted, because the decompression process restores their 0 value in the `udb` structure without needing any recorded information to do so. Each nonzero data field consists of an identifying token, a length, and a value. Compressed data can be copied but cannot be decompressed without the decompression algorithm in the access method library.

Format of User Entry

The format of a user entry as defined in the `/usr/include/udb.h` file is a property of `libudb`, and the interface is described in `libudb`(3C).

NOTES

You can find the external representation of a record in the user database in the `libudb.3c` file. To save space in the file, much of the information is packed as densely as is practical, using the structure defined in the library source file (`libc/gen/uentrydb.c`). Transformation functions within the library convert between external and internal representation for the caller.

FILES

`/etc/udb`	User information
`/etc/udb.public`	Public user information
`/etc/udb_2/udb.index`	Public extension index file
`/etc/udb_2/udb.priva`	Private field extension file
`/etc/udb_2/udb.pubva`	Public field extension file
`/usr/include/sharedefs.h`	File of reasons for eviction by the fair-share scheduler
`/usr/include/sys/secparm.h`	File of user permissions
`/usr/include/udb.h`	Structure definition of user database files

SEE ALSO

`acid`(5), `group`(5), `passwd`(5)

`udbsee`(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

`libudb`(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

`udbgen`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

updaters – Configuration file for NIS updating

SYNOPSIS

/etc/yp/updaters

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The /etc/yp/updaters file is a makefile (see make(1)) that updates network information services (NIS) databases. You can update databases only in a secure network; that is, a network that has a publickey(5) database. Each entry in the file is a make target for a particular NIS database. For example, if an NIS database named passwd.byname can be updated, a make target named passwd.byname should be in the updaters file with the appropriate command to update the file.

The requesting client passes the information necessary to make the update to the ypupdated(8) program, which passes the information to the update(3X) command through standard input. This information is described in the following list:

- Network name of client wanting to make the update (a string)
- Type of update (an integer)
- Number of bytes in key (an integer)
- Actual bytes of key
- Number of bytes in data (an integer)
- Actual bytes of data

Each of the items is followed by a newline character, except for actual bytes of key and actual bytes of data.

After getting this information through standard input, the command to update the particular database decides whether the user is allowed to make the change. If not, the command exits with the status of NISERR_ACCESS. If the user is allowed to make the change, the command makes the change and exits with a status of 0. If any errors exist that can prevent the updater from making the change, updaters exits with the status that matches a valid NIS error code described in the rpcsvc/ypclnt.h file.

FILES

/etc/yp/updaters	File that updates NIS databases

SEE ALSO

`publickey`(5)

`make`(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011,

`ypupdated`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

utmp, wtmp – utmp and wtmp file formats

SYNOPSIS

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

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The utmp and wtmp files hold user information for such commands as login(1), who(1), and write(1). Accounting programs such as csaline(8) and acctcon1(8) also process the utmp and wtmp files. These files have the following structure, as defined by the utmp.h include file:

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

struct utmp {
       char    ut_user[8];         /* User login name              */
       char    ut_id[4];           /* /etc/lines ID (usually line #)*/
       char    ut_line[12];        /* device name (console, lnxx)  */
       char    ut_host[24];        /* Name of remote machine       */
       short   ut_pid;             /* process ID                   */
       short   ut_type;            /* type of entry                */
       struct  exit_status  {
               short  e_termination; /* Process termination status  */
               short  e_exit;        /* Process exit status         */
       } ut_exit;                    /* Exit status of process      */
                                     /* marked as DEAD_PROCESS      */
       time_t  ut_time;              /* time entry was made         */
       char    ut_tpath[TPATHSIZ];   /* path of temporary file      */
       short   ut_jid;               /* job ID of pgrp leader       */
} ;
```

```
/*     Definitions for ut_type */
#define  EMPTY         0
#define  RUN_LVL       1
#define  BOOT_TIME     2
#define  OLD_TIME      3
#define  NEW_TIME      4
#define  INIT_PROCESS  5         /* Process spawned by "init"       */
#define  LOGIN_PROCESS 6         /* A process waiting for login     */
#define  USER_PROCESS  7         /* A user process                  */
#define  DEAD_PROCESS  8
#define  ACCOUNTING    9
#define  UTMAXTYPE  ACCOUNTING   /* Largest legal value of ut_type  */

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

FILES

`/etc/utmp`	File of user information
`/etc/wtmp`	File of user information
`/usr/include/sys/types.h`	Data type definition file
`/usr/include/utmp.h`	Format definition for the `utmp` and `wtmp` files

SEE ALSO

`last`(1B), `login`(1), `who`(1), `write`(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

`getut`(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

`acctcon1`(8), `csaline`(8), `fwtmp`(8), `wtmpfix`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

uuencode – Encoded uuencode file format

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

Files output by uuencode(1) consist of a header line, followed by a number of body lines, and a trailer line. The uudecode(1) command ignores any lines that precede the header or following the trailer. Lines preceding a header must not look like a header.

The header line is distinguished by having begin the first six characters. The word begin is followed by a mode (in octal) and a string that names the remote file. A space separates the three items in the header line.

The body consists of a number of lines, each consisting of at most 62 characters (including the trailing newline character). These consist of a character count, followed by encoded characters, followed by a newline character. The character count is one printing character, and it represents an integer, the number of bytes the rest of the line represents. Such integers are always in the range from 0 to 63 and can be determined by subtracting the character space (octal 40) from the character.

Groups of 3 bytes are stored in 4 characters, 6 bits per character. All are offset by a space to make the characters printing. The last line may be shorter than the normal 45 bytes. If the size is not a multiple of 3, this fact can be determined by the value of the count on the last line. Extra characters will be included to make the character count a multiple of 4. The body is terminated by a line with a count of 0. This line consists of one ASCII space.

The trailer line consists of end on a line by itself.

SEE ALSO

mail(1) for electronic message system information
uudecode(1) to decode a binary file
in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

NAME

values – Machine-dependent values definition file

SYNOPSIS

```
#include <values.h>
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The values.h include file contains a set of manifest constants defined for Cray Research processor architectures. The model assumed for integers is binary representation (twos complement), where the sign is represented by the value of the high-order bit.

The most important constants are defined as follows:

DSIGNIF Number of significant bits in the mantissa of a double-precision, floating-point number

FSIGNIF Number of significant bits in the mantissa of a single-precision, floating-point number

HIBITI Value of a regular integer with only the high-order bit set (usually the same as HIBITS or HIBITL)

HIBITL Value of a long integer with only the high-order bit set

HIBITS Value of a short integer with only the high-order bit set

MAXDOUBLE, LN_MAXDOUBLE
Maximum value of a double-precision, floating-point number and its natural logarithm

MAXFLOAT, LN_MAXFLOAT
Maximum value of a single-precision, floating-point number and its natural logarithm

MAXINT Maximum value of a signed regular integer (usually the same as MAXSHORT or MAXLONG)

MAXLONG Maximum value of a signed long integer

MAXSHORT Maximum value of a signed short integer

MINDOUBLE, LN_MINDOUBLE
Minimum positive value of a double-precision, floating-point number and its natural logarithm

MINFLOAT, LN_MINFLOAT
Minimum positive value of a single-precision, floating-point number and its natural logarithm

FILES

`/usr/include/values.h` Machine-dependent values definitions

SEE ALSO

`float.h`(3C), `numeric_lim`(3C), `values.h`(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

NAME

ypfiles – Network information service (NIS) database and directory structure

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The network information service (NIS) function provides a simple network look-up service that consists of databases and processes. The NIS network look-up service uses a database of dbm files in the /etc/yp directory.

A dbm database consists of two files created by calls to the dbm(3C) library package. One has the file name extension .pag and the other has the file name extension .dir. For instance, the database named hosts.byname is implemented by the pair of files names hosts.byname.pag and hosts.byname.dir.

A dbm database served by the NIS is called an NIS *map*. An NIS *domain* is a named set of NIS maps. Each NIS domain is implemented as a subdirectory of /etc/yp. Any number of NIS domains can exist; each may contain any number of maps.

The NIS look-up service itself requires no maps, although they may be required for the normal operation of other parts of the system. There is no list of maps that NIS serves; if the map exists in a given domain and a client asks about it, the NIS serves it. For a map to be accessible consistently, it must exist on all NIS servers for the domain. To provide data consistency between the replicated maps, make an entry to transfer the NIS map periodically from each NIS server (with the ypxfr(8) command) /usr/lib/crontab on each server.

NIS maps should contain key-value pairs that consist of the YP_LAST_MODIFIED key and the YP_MASTER_NAME key. YP_LAST_MODIFIED is the order number or time (in seconds) when the map was built; its value is a 10-character ASCII number. YP_MASTER_NAME is the name of the NIS master server. The makedbm(8) command generates the key-value pairs automatically. NIS can serve a map that does not contain key-value pairs, but the ypserv(8) process cannot return values for a Get_order_number or Get_master_name request. When ypxfr(8) transfers a map from a master NIS server to a slave, ypxfr(8) also uses the values of these two keys.

You must generate and modify NIS maps only at the master server. To copy them to the slaves, use ypxfr(8). This prevents potential byte-ordering problems among NIS servers running on machines that have different architectures and reduces the amount of disk space required for the dbm files. To set up the NIS database initially for both masters and slaves, use ypinit(8).

After the server databases are set up, the contents of some maps probably will change. Generally, an ASCII source version of the database exists on the master. To change this version, use a text editor. The edited copy is incorporated into the NIS map and is propagated from the master to the slaves by running the `/etc/yp/yp.mk` makefile. All standard maps have entries in `/etc/yp/yp.mk`; if you add an NIS map, edit this file to support the new map. The makefile uses `makedbm`(8) to generate the NIS map on the master and `yppush`(8) to propagate the changed map to the slaves. The `yppush`(8) command is a client of the map `ypservers`, which lists all the NIS servers.

NOTES

The NIS was formerly known as yellow pages, which explains the `yp`-prefix on command and directory names.

SEE ALSO

`makedbm`(8), `rpcinfo`(8), `ypinit`(8), `yppoll`(8), `yppush`(8), `ypserv`(8), `ypxfr`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

intro – Miscellaneous information pages

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

Section 7D contains miscellaneous documentation, mostly concerning DWB. DWB, which is based on AT&T's Documenter's Workbench, runs under UNICOS and provides a variety of macro packages.

In addition to DWB man pages, this section includes a man page for msg(7D), the text formatting macros for UNICOS messages.

NAME

eqnchar – Special character definitions for eqn(1)

SYNOPSIS

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

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

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The eqnchar command contains troff(1) and nroff(1) character definitions for constructing characters that are not available on the Graphic Systems typesetter. These definitions are primarily intended for use with eqn(1) and neqn(1). It contains definitions for the following characters:

ciplus	⊕	| |	‖	square	□
citimes	⊗	langle	⟨	circle	○
wig	~	rangle	⟩	blot	■
-wig	≃	hbar	ℏ	bullet	•
>wig	≳	ppd	⊥	prop	∝
<wig	≲	<->	↔	empty	∅
=wig		<=>		member	∈
star	*	|<	≮	nomem	∉
bigstar	*	|>	≯	cup	∪
=dot	≐	ang	∠	cap	∩
orsign	∨	rang	⌊	incl	⊆
andsign	∧	3dot	⋮	subset	⊂
=del	≜	thf	∴	supset	⊃
oppA	∀	quarter	¼	!subset	⊈
oppE	∃|	3quarter	¾	!supset	⊉
angstrom	Å	degree	°		

FILES

/usr/pub/eqnchar

SEE ALSO

eqn(1), nroff(1), troff(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

NAME

man – Macros to format AT&T reference manual pages

SYNOPSIS

nroff -man *filename* ...

troff -man *filename* ...

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

These macros are used to lay out the reference pages in this manual. If *filename* contains format input for a preprocessor, the preceding commands must be piped through the appropriate preprocessor. man(1) handles this automatically. See the Conventions subsection.

Any text argument *t* may be 0 to 6 words. You may use quotation marks to include SPACE characters in a word. If *text* is empty, the special treatment is applied to the next input line with text to be printed. In this way, you may use .I to italicize a whole line, or use .SB to make small bold letters.

A prevailing indent distance is remembered between successive indented paragraphs, and it is reset to default value on reaching a nonindented paragraph. Default units for indents *i* are ens.

Type font and size are reset to default values before each paragraph, and after processing font and size setting macros.

These strings are predefined by -man:

*R ‘ ’, ‘(Reg)’ in nroff

*S Change to default type size.

Requests

n.t.l. Next text line

p.i. Prevailing indent

Request	Cause break	If no argument	Explanation
.B *t*	No	*t*=n.t.l.	Text is in bold font.
.BI *t*	No	*t*=n.t.l.	Join words, alternating bold and italic.
.BR *t*	No	*t*=n.t.l.	Join words, alternating bold and roman.
.DT	No	.5i 1i...	Restore default tabs.

Request	Cause break	If no argument	Explanation
`.HP` *i*	Yes	*i*=p.i.	Begin paragraph with hanging indent. Set prevailing indent to *i*.
`.I` *t*	No	*t*=n.t.l.	Text is italic.
`.IB` *t*	No	*t*=n.t.l.	Join words, alternating italic and bold.
`.IP` *x i*	Yes	*x*=""	Same as `.TP` with tag *x*.
`.IR` *t*	No	*t*=n.t.l.	Join words, alternating italic and roman.
`.IX` *t*	No	–	Index macro, for Sun internal use.
`.LP`	Yes	–	Begin left-aligned paragraph. Set prevailing indent to 0.5i.
`.PD` *d*	No	*d*=.4v	Set vertical distance between paragraphs.
`.PP`	Yes	–	Same as `.LP`.
`.RE`	Yes	–	End of relative indent. Restore prevailing indent.
`.RB` *t*	No	*t*=n.t.l.	Join words, alternating roman and bold.
`.RI` *t*	No	*t*=n.t.l.	Join words, alternating roman and italic.
`.RS` *i*	Yes	*i*=p.i.	Start relative indent, increase indent by *i*. Set prevailing indent to 0.5i for nested indents.
`.SB` *t*	No	–	Reduce size of text by 1 point and make text bold.
`.SH` *t*	Yes	–	Section heading.
`.SM` *t*	No	*t*=n.t.l.	Reduce size of text by 1 point.
`.SS` *t*	Yes	*t*=n.t.l.	Section subheading.
`.TH` *n s d f m*	Yes	–	Begin reference page *n*, of section *s*; *d* is the date of the most recent change. If present, *f* is the left page footer; *m* is the main page (center) header. Set prevailing indent and tabs to 0.5i.
`.TP` *i*	Yes	*i*=p.i.	Begin indented paragraph, with the tag given on the next text line. Set prevailing indent to *i*.
`.TX` *t p*	No	–	Resolve the title abbreviation *t*; join to punctuation mark (or text) *p*.

Conventions

When formatting a man page, `man`(1) examines the first line to determine whether it requires special processing. For example, a first line consisting of the following code indicates that the man page must be run through the `tbl`(1) preprocessor:

```
'\" t
```

A typical manual page for a command or function is laid out as follows:

`.TH` *title* [1–8]
: The name of the command or function, which serves as the title of the manual page. This is followed by the number of the section in which it appears.

`.SH NAME`
: The name, or list of names, by which the command is called, followed by a dash and then a one-line summary of the action performed. All in roman font, this section contains no `troff`(1) commands or escapes, and no macro requests. It is used to generate the `whatis`(1) database.

`.SH SYNOPSIS`

Commands
: The syntax of the command and its arguments, as typed on the command line. When in bold, you must type a word exactly as printed. When in italics, you can replace a word with an argument. References to bold or italicized items are not capitalized in other sections, even when they begin a sentence.

  Syntactic symbols appear in roman face:

  `[ ]` An argument, when surrounded by brackets, is optional.

  `|` Arguments separated by a vertical bar are exclusive. You can supply only one item from such a list.

  `...` Arguments followed by an ellipsis can be repeated. When an ellipsis follows a bracketed set, you can repeat the expression within the brackets.

Functions
: If required, the data declaration, or `#include` directive, is shown first, followed by the function declaration. Otherwise, the function declaration is shown.

`.SH DESCRIPTION`
: A narrative overview of the command or function's external behavior. This includes how it interacts with files or data, and how it handles the standard input, standard output, and standard error. Internals and implementation details are usually omitted. This section tries to provide a succinct overview.

  Literal text from the synopsis appears in constant width, as do literal file names and references to items that appear elsewhere in the reference manuals.

  Arguments are italicized. If a command interprets either subcommands or an input grammar, its command interface or input grammar is usually described in a USAGE section, which follows the OPTIONS section. The DESCRIPTION section describes only the behavior of the command itself, not that of subcommands.

`.SH OPTIONS`
: The list of options, along with a description of how each affects the command's operation.

`.SH FILES`
: A list of files associated with the command or function.

`.SH SEE ALSO`
: A comma-separated list of related man pages, followed by references to other published materials.

`.SH DIAGNOSTICS`
: A list of diagnostic messages and an explanation of each.

`.SH BUGS`
: A description of limitations, known defects, and possible problems associated with the command or function.

FILES

`/usr/lib/tmac/tmac.an`

SEE ALSO

`man`(1), `nroff`(1), `tbl`(1), `troff`(1), `whatis`(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

NAME

me – Macros for formatting papers

SYNOPSIS

nroff -me [*options*] *filename* ...

troff -me [*options*] *filename* ...

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The me package of nroff(1) and troff(1) macro definitions provides a canned formatting facility for technical papers in various formats. When producing two-column output on a terminal, filter the output through col(1).

A definition of the macro requests follow. Many nroff(1) and troff(1) requests are unsafe in conjunction with this package; however, you may use these requests with impunity after the first .pp:

.bp	Begins new page.
.br	Breaks output line here.
.sp *n*	Inserts *n* spacing lines.
.ls *n*	(line spacing) *n*=1 single; *n*=2 double-space.
.na	Does not align right margin.
.ce *n*	Centers next *n* lines.
.ul *n*	Underlines next *n* lines.
.sz +*n*	Adds *n* to point size.

Output of the eqn(1), neqn(1), and tbl(1) preprocessors for equations and tables is acceptable as input.

Requests

In the following list, *initialization* refers to the first .pp, .lp, .ip, .np, .sh, or .uh macro. This list is incomplete.

Request	Initial value	Cause break	Explanation
.(c	–	Yes	Begins centered block.
.(d	–	No	Begins delayed text.
.(f	–	No	Begins footnote.

Request	Initial value	Cause break	Explanation
.(l	–	Yes	Begins list.
.(q	–	Yes	Begins major quote.
.(x*x*	–	No	Begins indexed item in index *x*.
.(z	–	No	Begins floating keep.
.)c	–	Yes	Ends centered block.
.)d	–	Yes	Ends delayed text.
.)f	–	Yes	Ends footnote.
.)l	–	Yes	Ends list.
.)q	–	Yes	Ends major quote.
.)x	–	Yes	Ends index item.
.)z	–	Yes	Ends floating keep.
.++ *m H*	–	No	Defines paper section. *m* defines the part of the paper, and it can be C (chapter), A (appendix), P (preliminary, for instance, abstract, table of contents, and so on), B (bibliography), RC (chapters renumbered from page one each chapter), or RA (appendix renumbered from page 1).
.+c *T*	–	Yes	Begins chapter (or appendix, and so on, as set by .++). *T* is the chapter title.
.1c	1	Yes	One-column format on a new page.
.2c	1	Yes	Two-column format.
.EN	–	Yes	Space after equation produced by eqn(1) or meqn.
.EQ *x y*	–	Yes	Precedes equation; break out and add space. Equation number is *y*. The optional argument *x* may be *I* to indent equation (default), *L* to left-adjust the equation, or *C* to center the equation.
.GE	–	Yes	Ends *gremlin* picture.
.GS	–	Yes	Begins *gremlin* picture.
.PE	–	Yes	Ends pic(1) picture.
.PS	–	Yes	Begins pic(1) picture.
.TE	–	Yes	Ends table.

Request	Initial value	Cause break	Explanation
`.TH`	–	Yes	Ends heading section of table.
`.TS` *x*	–	Yes	Begins table; if *x* is *H*, table has repeated heading.
`.ac` *A N*	–	No	Sets up for ACM style output. *A* is the Author's name(s), and *N* is the total number of pages. Must be given before the first initialization.
`.b` *x*	No	No	Prints *x* in bold face; if no argument, switches to bold face.
`.ba` *+n*	0	Yes	Augments the base indent by *n*. Use this indent to set the indent on regular text (such as paragraphs).
`.bc`	No	Yes	Begins new column.
`.bi` *x*	No	No	Prints *x* in bold italics (no-fill only).
`.bu`	–	Yes	Begins bulleted paragraph.
`.bx` *x*	No	No	Prints *x* in a box (no-fill only).
`.ef` *'x'y'z*	`''''`	No	Sets even footer to *x y z*.
`.eh` *'x'y'z*	`''''`	No	Sets even header to *x y z*.
`.fo` *'x'y'z*	`''''`	No	Sets footer to *x y z*.
`.hx`	–	No	Suppresses headers and footers on next page.
`.he` *'x'y'z*	`''''`	No	Sets header to *x y z*.
`.hl`	–	Yes	Draws a horizontal line.
`.i` *x*	No	No	Italicizes *x*; if *x* missing, italic text follows.
`.ip` *x y*	No	Yes	Starts indented paragraph, with hanging tag *x*. Indentation is *y* ens (default 5).
`.lp`	Yes	Yes	Starts left-blocked paragraph.
`.lo`	–	No	Reads in a file of local macros of the form `.*`*x*. Must be given before initialization.
`.np`	1	Yes	Starts numbered paragraph.
`.of` *'x'y'z*	`''''`	No	Sets odd footer to *x y z*.
`.oh` *'x'y'z*	`''''`	No	Sets odd header to *x y z*.
`.pd`	–	Yes	Prints delayed text.
`.pp`	No	Yes	Begins paragraph. First line is indented.

Request	Initial value	Cause break	Explanation
`.r`	Yes	No	Roman text follows.
`.re`	–	No	Resets tabs to default values.
`.sc`	No	No	Reads in a file of special characters and diacritical marks. Must be given before initialization.
`.sh` *n x*	–	Yes	Section head follows, font automatically bold. *n* is level of section, and *x* is title of section.
`.sk`	No	No	Leaves the next page blank. Only one page is remembered ahead.
`.sm` *x*	–	No	Sets *x* in a smaller point size.
`.sz` +*n*	10p	No	Augments the point size by *n* points.
`.th`	No	No	Produces the paper in thesis format. Must be given before initialization.
`.tp`	No	Yes	Begins title page.
`.u` *x*	–	No	Underlines argument (even in `troff`(1)). (No-fill only).
`.uh`	–	Yes	Like `.sh`, but unnumbered.
`.xp` *x*	–	No	Prints index *x*.

FILES

`/usr/lib/tmac/*.me`

`/usr/lib/tmac/e`

SEE ALSO

`col`(1), `eqn`(1), `nroff`(1), `pic`(1), `tbl`(1), `troff`(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

NAME

ms – Text formatting macros

SYNOPSIS

nroff -ms [*options*] *filename* ...

troff -ms [*options*] *filename* ...

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The ms package of nroff(1) and troff(1) macro definitions provides a formatting facility for various styles of articles, theses, and books. When producing two-column output on a terminal or line printer, or when reverse-line motions are needed, filter the output through col(1). Definitions of all external -ms macros follow.

NOTE: This -ms macro package is an extended version written at Berkeley and is a superset of the standard -ms macro packages as supplied by Bell Labs. Some of the Bell Labs macros have been removed; for instance, it is assumed that the user has little interest in producing headers stating that the memo was generated at Whippany Labs.

Many nroff(1) and troff(1) requests are unsafe in conjunction with this package. However, you may use the first four requests that follow with impunity after initialization, and you may use the last two even before initialization:

.bp	Begins new page.
.br	Breaks output line.
.sp *n*	Inserts *n* spacing lines.
.ce *n*	Centers next *n* lines.
.ls *n*	Line spacing: *n*=1 single; *n*=2 double-space.
.na	Does not align right margin.

Font and point size changes with \f and \s also are allowed (for example, \fI*word*\fR italicizes *word*). Output of the tbl(1) and eqn(1) preprocessors for equations and tables is acceptable as input.

Requests

Macro name	Initial value	Break? Reset?	Explanation
.AB x	–	y	Begins abstract; if *x* = no, do not label abstract.
.AE	–	y	Ends abstract.
.AI	–	y	Author's institution.
.AM	–	n	Better accent mark definitions.
.AU	–	y	Author's name.
.B *x*	–	n	Emboldens *x*; if no *x*, switches to bold face.
.B1	–	y	Begins text to be enclosed in a box.
.B2	–	y	Ends boxed text and prints it.
.BT	date	n	Bottom title, printed at foot of page.
.BX *x*	–	n	Prints word *x* in a box.
.CM	if t	n	Cuts mark between pages.
.CT	–	y,y	Chapter title: page number moved to CF (TM only).
.DA *x*	if n	n	Forces date *x* at bottom of page; today if no *x*.
.DE	–	y	Ends display (unfilled text) of any kind.
.DS *Y*	I	y	Begins display with keep; *x* = I, L, C, B; *y* = indent.
.ID*z*	8n,.5i	y	Indented display with no keep; *y* = indent.
.LD	–	y	Left display with no keep.
.CD	–	y	Centered display with no keep.
.BD	–	y	Block display; centers entire block.
.EF *x*	–	n	Even page footer *x* (three part as for .tl).
.EH *x*	–	n	Even page header *x* (three part as for .tl).
.EN	–	y	Ends displayed equation produced by eqn(1).
.EQ *Y*	–	y	Breaks out equation; *x* = L,I,C; *y* = equation number.
.FE	–	n	Ends footnote to be placed at bottom of page.
.FP	–	n	Numbered footnote paragraph; may be redefined.
.FS *x*	–	n	Starts footnote; *x* is optional footnote label.
.HD	undef	n	Optional page header below header margin.
.I *x*	–	n	Italicizes *x*; if no *x*, switches to italics.
.IP *Y*	–	y,y	Indented paragraph, with hanging tag *x*; *y* = indent.
.IX *Y*	–	y	Indexes words *x y* and so on (up to five levels).
.KE	–	n	Ends keep of any kind.
.KF	–	n	Begins floating keep; text fills remainder of page.
.KS	–	y	Begins keep; unit kept together on a single page.

Macro name	Initial value	Break? Reset?	Explanation
.LG	–	n	Larger; increases point size by 2.
.LP	–	y,y	Left (block) paragraph.
.MC *x*	–	y,y	Multiple columns; *x* = column width.
.ND *x*	if t	n	No date in page footer; *x* is date on covers.
.NH *Y*	–	y,y	Numbered header; *x* = level, *x* = *0* resets, *x* = S sets to *y*.
.NL	10p	n	Sets point size back to normal.
.OF *x*	–	n	Odd page footer *x* (three part as for .tl).
.OH *x*	–	n	Odd page header *x* (three part as for .tl).
.P1	if TM	n	Prints header on first page.
.PP	–	y,y	Paragraph with first line indented.
.PT	- % -	n	Page title, printed at head of page.
.PX *x*	–	y	Prints index (table of contents); *x* = no suppresses title.
.QP	–	y,y	Quotes paragraph (indented and shorter).
.R	on	n	Returns to roman font.
.RE	5n	y,y	Retreats: ends level of relative indentation.
.RP *x*	–	n	Released paper format; *x* = no stops title on first page.
.RS	5n	y,y	Right shifts: starts level of relative indentation.
.SH	–	y,y	Section header, in bold face.
.SM	–	n	Smaller; decreases point size by 2.
.TA	8n,5n	n	Sets TAB characters to 8n 16n ... (nroff(1)) 5n 10n ... (troff(1)).
.TC *x*	–	y	Prints table of contents at end; *x* = no suppresses title.
.TE	–	y	Ends table processed by tbl(1).
.TH	–	y	Ends multipage header of table.
.TL	–	y	Title in bold face and two points larger.
.TM	off	n	UC Berkeley thesis mode.
.TS *x*	–	y,y	Begins table; if *x* = H, table has multipage header.
.UL *x*	–	n	Underlines *x*, even in troff(1).
.UX *x*	–	n	UNIX; trademark message first time; *x* appended
.XA *Y*	–	y	Another index entry; *x* = page or no for none; *y* = indent.
.XE	–	y	Ends index entry (or series of .IX entries).
.XP	–	y,y	Paragraph with first line exdented, others indented.
.XS *Y*	–	y	Begins index entry; *x* = page or no for none; *y* = indent.
.1C	on	y,y	One-column format, on a new page.

Macro name	Initial value	Break? Reset?	Explanation
.2C	–	y,y	Begins two-column format.
.]-	–	n	Beginning of refer reference.
.[0	–	n	Ends unclassifiable type of reference.
.[N	–	n	N = 1:journal-article, 2:book, 3:book-article, 4:report.

Registers

To control formatting distances in -ms, use built-in number registers. For example, the following command line sets the line length to 6.5 inches:

```
.nr  LL  6.5i
```

A table of number registers and their default values follows:

Name	Register controls	Takes effect	Default
PS	Point size	Paragraph	10
VS	Vertical spacing	Paragraph	12
LL	Line length	Paragraph	6i
LT	Title length	Next page	Same as LL
FL	Footnote length	Next .FS	5.5i
PD	Paragraph distance	Paragraph	1v (if n), 0.3v (if t)
DD	Display distance	Displays	1v (if n), 0.5v (if t)
PI	Paragraph indent	Paragraph	5*n*
QI	Quote indent	Next .QP	5*n*
FI	Footnote indent	Next .FS	2*n*
PO	Page offset	Next page	0 (if n), ~1i (if t)
HM	Header margin	Next page	1i
FM	Footer margin	Next page	1i
FF	Footnote format	Next .FS	0 (1, 2, 3 available)

When resetting these values, make sure to specify the appropriate units. Setting the line length to 7, for example, results in output with one character per line; setting FF to 1 suppresses footnote superscripting; setting it to 2 also suppresses indentation of the first line; and setting it to 3 produces an .IP-like footnote paragraph.

A list of string registers available in -ms follows; you may use them anywhere in the text:

Name	String's Function
*Q	Quote (" in nroff(1), `` in troff(1))
*U	Unquote (" in nroff(1), '' in troff(1))
*–	Dash (-- in nroff(1), — in troff(1))
*(MO	Month (month of the year)
*(DY	Day (current date)
**	Automatically numbered footnote
*´	Acute accent (before letter)
*`	Grave accent (before letter)
*^	Circumflex (before letter)
*,	Cedilla (before letter)
*:	Umlaut (before letter)
*~	Tilde (before letter)

When using the extended accent mark definitions available with .AM, these strings should come after, rather than before, the letter to be accented.

BUGS

Floating keeps and regular keeps are diverted to the same space; therefore, you cannot mix them together with predictable results.

FILES

/usr/lib/tmac/ms.???

/usr/lib/tmac/s

SEE ALSO

col(1), eqn(1), nroff(1), tbl(1), troff(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

NAME

msg – Text formatting macros for UNICOS messages

SYNOPSIS

```
nroff -msg files
troff -msg files
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The msg macros are a package of nroff and troff macro definitions that provides a formatting facility for the printed documents of the UNICOS message system. The Requests subsection defines all available macros.

Virtually all nroff and troff directives should be unnecessary in conjunction with this macro package. However, they are available if desired and should work as documented. You also can create tables and equations by using tbl(1) and eqn(1) directives, respectively; these processors work with the msg macros.

For a description of how to format and print a file that uses the msg macros, see the *Cray Message System Programmer's Guide*, Cray Research publication SG–2121.

Requests

Unless otherwise noted, all msg requests (macros) must start at the beginning of a line. No other text lines or words can start with a . symbol.

.2S — Starts two-column mode. This macro automatically makes point size equal to 9 and vertical spacing (leading) equal to 11.

.2E — Ends two-column mode.

.BL [*x*] — (Bulleted list) Makes an entry in a bulleted list. The *x* is either d for double-spaced list or s for single-spaced list (the default).

.CF "*string*" — (Center footer string) Defines *string* as the center string for footers. In Cray Research style, the center footer contains the security level of the manual (public, private, or proprietary).

.CH *x* "*string1*" "*string2*" — (Column headings) Makes underlined column heads for two-column lists; *x* is indent (as in _TL); *string1* and *string2* are column heads. *x* cannot be less than 1.28 or greater than 47. If *x* is less than the width of *string1*, the width of *string1* is used as the first column width. If the first column width would leave the second column less than 5-ens wide, *x* is adjusted to keep the second column 5-ens wide.

`.CR` [*x*] [*y*] (Counter reset) Resets the counter that is output by the `\*n` string. The *x* argument is the number at which the next count will start (default is 1). The *y* argument is the type of the counter character (default is Arabic numerals). For a list of the values that you may use for *y*, see the description of the *x* argument of the `.NL` macro).

`.CS` (Code start) Begins a block of code in Courier font. In two-column mode (point size 9), lines between `.CS` and `.CE` macros cannot consist of more than 44 characters.

`.CE` (Code end) Ends a code block started with `.CS`.

`.DL` [*x*] (Dashed list) Makes an entry in a dashed list. The *x* is either `d` for double-spaced list or `s` for single-spaced list (the default).

`.EQ` Starts equation (with `eqn`); ends equation with `.EN`.

`.EN` Ends equation (with `eqn`).

`.GC "`*group*"

(Group code) Defines the group code to be used for a set of messages. You must define the group code because it is printed as part of the message identifier for each message.

`.KT` *x*[`i`] (Keep together) Keeps the next *x* output lines or the next *x* inches from breaking over a page. *x* is interpreted as a line count unless you specify the `i` suffix. In that case, it is interpreted as a number of inches (`3.9i`, for example). If *x* is more than 53, the `_KT` macro is ignored.

`.MN` "*string*" (Manual number) Defines the manual (publication) number for page footers as *string*.

`.MS` *msg#* [`b`]

This macro is substituted automatically for `$nexp` tags by the `catxt`(1) command.

(Message start) Starts a message block that, by default, will not break over columns or pages, unless it is longer than one column (page). To force a message to break over a column or page, use the `b` option. The *msg-#* is the message number used in the online message system. If the text will be used in the UNICOS message system, this argument is required.

`.ME` (Message end) Ends a message block.

`.MT` "*string*" (Manual title) Defines the manual title for use in page headers as *string*.

`.NL` [*x* [*y* [*z* [`d`]]]]

(Numbered list) Makes an entry in a numbered list. The *x* is either the type of numerals you want (default is Arabic), or a `d` to specify a double-spaced, Arabic-numbered list. The *y* is the indent between the numerals and the paragraph. The *z* is the number at which to (re)start the count if you want something other than the first character in the series (1 or i or A, and so on). If you want a double-spaced list that uses something other than Arabic numerals (so that you cannot specify `d` for *x*), specify the `d` as a fourth argument to `.NL`. The *x* argument can have one of the following values:

Value	Default indent	Result
1	3.3	Arabic numerals (the default)
a	3.3	Lowercase letters
A	4	Uppercase letters
i	4.5	Lowercase roman numerals
I	5	Uppercase roman numerals
d	3.3	Arabic numerals, with full blank lines between list entries

You should end numbered lists with the .NN macro.

.NN (Numbered-list end) Ends numbered list (resets numbers to 1 at that level of indent).

.PP [*x*] (Paragraph (resets indent)) *x* is the number of (printed) lines to keep together on the same page; the default is 4. Use this argument only if you use the b option to .MS or if your message is longer than one column.

.RN [*fig-no* [*pg-no* [*tbl-no* [*sec-no* [*sec-style*]]]]
(Renumber) Placed at the head of a section you want to print by itself (without preceding sections), this macro starts figure, page, table, and section numbers at the values specified. The last argument is either 0 for numeric section numbers (the default) or A for alphabetic section numbers (used in appendixes).

.SN (Sequential numbering) Ensures that pages, figures, and tables are numbered sequentially across multiple files; specify as the last line of each file. Also, allows multiple sections in the same file; put just before any .ST macros other than the very first one in a file.

.SP [*x*] Adds vertical space (leading) without resetting indentation. Use _SP instead of _PP in indented lists and examples). The *x* is the number of following lines to keep in one block (not break over pages).

.SQ [*x*] Space half a line (use instead of _PP in indented lists and examples). The *x* is the number of following lines to keep in one block (not break over pages).

.ST "*string*" (Section title) *string* is the section title.

.TL [*x* [*y*]] (Tagged list) Makes a tagged-paragraph list. The following line is the tag, and, on only the first entry, *x* is the indent; it cannot be less than 1.28 or greater than 47, and if you do not specify it, it defaults to 5 ens. The *y* is either d for double-spaced list or s for single-spaced list (the default).

.TS (Table start) Begins a table to be processed with tbl. Be sure that tables are 3.3 i. or narrower in width.

.TE (Table end) Ends a table to be processed by tbl.

Font Changes

`\fB`	Change to New Century bold font (can start anywhere on a line).
`\fI` or `\*V`	Change to New Century italic font (can start anywhere on a line).
`\fR`	Change to New Century roman font (can start anywhere on a line).
`\*C`	Change to Courier font (can start anywhere on a line).
`\*(Cb` or `\f(CB`	Change to Courier bold font (can start anywhere on a line).
`\*(Co` or `\f(CO`	Change to Courier italic font (can start anywhere on a line).
`\fP`	Change to previous font (use to undo font change).

Nonprinting Comments

`´\"`	Comment line; entire line is ignored when formatted (this version is preferred).

Predefined Strings

Hardware Names

`\*y`	`\%CRAY\ Y-MP`; the resulting text, "CRAY Y-MP", will not break over lines.
`\*(ys`	`\%CRAY\ Y-MP\ EL`; the resulting text, "CRAY Y-MP EL", will not break over lines.
`\*(EL`	`\%EL\ series`; the resulting text, "EL series", will not break over lines.
`\*(EO`	`\%EL\ IOS`; the resulting text, "EL IOS", will not break over lines.
`\*(c9`	`\%CRAY\ C90`; the resulting text, "CRAY C90", will not break over lines.
`\*(Ie`	`\%IOS-E`; the resulting text, "IOS-E", will not break over lines.
`\*(IE`	`\%IOS model\ E`; the resulting text, "IOS model E", will not break over lines.
`\*m`	`\%CRAY\ T3D`; the resulting text, "CRAY T3D", will not break over lines.
`\*(MP`	`\%Cray\ MPP\ systems`; the resulting text, "Cray MPP systems", will not break over lines

Miscellaneous:

`\*(Ca`	CRI
`\*(Cr`	Cray Research, Inc.
`\*(UM`	`\%UNICOS\ MAX`; the resulting text, "UNICOS MAX", will not break over lines
`\*u`	UNICOS

FILES

`/usr/lib/tmac/tmac.sg` Message macro package

SEE ALSO

`catxt`(1), `explain`(1) describe UNICOS message system user commands
`eqn`(1), `nroff`(1), `tbl`(1), `troff`(1) describe text formatting utilities
in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

Cray Message System Programmer's Guide, Cray Research publication SG–2121, contains details about all aspects of the message system