CLIX™

System Administrator's Reference Manual

INTERGRAPH

SE

CLIX™

System Administrator's Reference Manual

INTERGRAPH

DSYS18310

CLIX System Administrator's Reference Manual

January 1990

DSYS18312
Edition 1

Warranties and Liabilities

All warranties given by Intergraph Corporation about equipment or software are set forth in your purchase contract.

The information and the software discussed in this document are subject to change without notice and should not be considered commitments by Intergraph Corporation.
Intergraph Corporation assumes no responsibility for any errors that may appear in this document.

The software discussed in this document is furnished under a license and may be used or copied only in accordance with the terms of this license.

No responsibility is assumed by Intergraph for the use or reliability of software on equipment that is not supplied by Intergraph or its affiliated companies.

Trademarks

Intergraph is a registered trademark of Intergraph Corporation.
InterPro, InterAct, and InterView are registered trademarks of Intergraph Corporation.
CLIX, InterServe, and CLIPPER are trademarks of Intergraph Corporation.

Other brands and product names are trademarks of their respective owners.

Classifications

This equipment is designed to comply with the requirements in Part 15 of the FCC rules for a class A computing device.

Copyrights

Intergraph Corporation
Huntsville, AL 35894–0001

Additional References

The following UNIX System V documentation is required reference material. These documents can be purchased individually or in sets from Intergraph:

Title	Release V.3
AT&T UNIX System V User's Reference Manual	DSYS08110
AT&T UNIX System V User's Reference Addendum	DSYS19410
AT&T UNIX System V Administrator's Reference Manual	DSYS08310
AT&T UNIX System V Administrator's Reference Addendum	DSYS19710
AT&T UNIX System V Programmer's Reference Manual	DSYS08510
AT&T UNIX System V Programmer's Reference Addendum	DSYS19510

The following UNIX System V documentation is suggested reference material. The following documents can be purchased individually or in sets from Intergraph:

Title	Release V.3
AT&T UNIX System V User's Guide	DSYS08010
AT&T UNIX System V Programming Guide	DSYS08410
AT&T UNIX System V Administrator's Guide	DSYS08210

Ordering Information

To order any of these documents:

- Within the United States contact your Customer Engineer or Sales Account Representative.
- For International locations, contact the Intergraph subsidiary or distributor where you purchased your workstation.

Support Information

If you have trouble with the workstation/server or the procedures described in this guide, contact Intergraph Customer Support at 1–800–633–7248. International customers should contact the Intergraph subsidiary or distributor where the workstation was purchased.

Introduction

Introduction

The *CLIX System Administrator's Reference Manual* describes the commands and special interfaces used by those who administer a CLIX system running on an Intergraph workstation or server.

This manual supplements the AT&T UNIX System V documentation and includes only additions and changes to System V found in the CLIX System.

The following documents provide related information:

- The *CLIX Programmer's & User's Reference Manual* describes the commands that constitute the basic software running on an Intergraph workstation or server, as well as system calls, library routines, file formats, and miscellaneous facilities.
- The *CLIX System Guide* contains procedures and tutorials designed to give instructions in how to perform tasks and background information about when and why these tasks are desirable.

The *CLIX System Administrator's Reference Manual* is divided into the following sections:

(1M) System Administrator Commands

(7) Special Interfaces

(7S) Special Files

(7B) BSD Network Interfaces

(7A) Asynchronous Interfaces

The *CLIX Programmer's & User's Reference Manual* is divided into the following sections:

(1) Commands

(2) System Calls

(2B) BSD System Calls

(2I) Intergraph System Calls

(3) Library Routines

(3C) and (3S) C Programming Language Utilities

(3B) BSD Library Routines

(3N) Intergraph Network Library Routines

(3R) RPC/XDR/YP Library Routines

(3A) Intergraph Synchronous/Asynchronous Library Routines

(4) File Formats

(5) Miscellaneous

The *CLIX System Guide* is divided into the following sections:

Part 1: System Administrator's Tutorials

1. FFS Tutorial
2. FFS Check Tutorial
3. BSD LP Spooler Tutorial
4. NQS Tutorial
5. YP Tutorial

Part 2: System Administrator's Procedures

1. System Rebuild
2. New Product Delivery
3. System Reconfiguration
4. FFS Installation
5. BSD Network Configuration
6. NFS/YP Installation
7. NQS Installation

Part 3: Programmer's & User's Tutorials

1. Technical Programming Tutorial
2. PROC Debugging Tutorial
3. Network Programming Tutorial
4. BSD Porting Tutorial
5. Introductory Socket Tutorial
6. Advanced Socket Tutorial
7. NQS Tutorial
8. RCS Tutorial
9. RPC/XDR Tutorial

References

Throughout this manual, numbers following a command are intended for easy cross-reference.

- Look up references followed by (1M), (7S), (7B), or (7A) in this document.
- Look up references followed by (1), (2B), (2I), (3C), (3B), (3N), (3R), (3A), (4), or (5) in the *CLIX Programmer's & User's Reference Manual.*
- Look up all other references in the appropriate CLIX document.

If the references are not in the CLIX document, refer to the appropriate UNIX System V manual.

Format

Most sections begin with a page labeled *intro*. Entries following the *intro* page are arranged alphabetically and may consist of more than one page. Some entries describe several routines, commands, etc. In such cases, the entry appears only once, alphabetized under its "primary" name. (An example of such an entry is *mount*(1M) which also describes the *umount* command.) To learn which manual page describes a secondary command, locate its name in the middle column of the "Permuted Index" and follow across that line to the name of the manual page listed in the right column.

All entries are based on a common format, but each part appears only where applicable:

- **NAME** gives the name(s) of the entry and briefly states its purpose.
- **SYNOPSIS** summarizes the use of the program being described. A few conventions are used, particularly in Section (1M) (*Commands*):
  - **Boldface** strings are literals and are to be typed just as they appear.
  - *Italic* strings usually represent substitutable argument and program names found elsewhere in the manual.
  - Brackets **[ ]** around an argument indicate that the argument is optional.
  - Braces { } around arguments indicate that one of the arguments should be chosen.
  - Ellipses **...** are used to show that the previous argument may be repeated.
- **DESCRIPTION** provides an overview of the command.
- **EXAMPLES** gives examples of usage, where appropriate.
- **FILES** gives the file names that are built into the program.
- **SEE ALSO** offers pointers to related information.
- **DIAGNOSTICS** discusses the diagnostic indications that may be produced. Messages that are intended to be self-explanatory are not listed.
- **NOTES** gives information that may be helpful under the particular circumstances described.
- **WARNINGS** points out potential pitfalls.
- **BUGS** gives known bugs and sometimes deficiencies.
- **CAVEATS** gives details of the implementation that might affect usage.

- **IDENTIFICATION** gives the author of the program.

Table of Contents & Permuted Index

Preceding Section (1M) is a "Table of Contents" (listing both primary and secondary command entries) and a "Permuted Index." Each line of the "Table of Contents" contains the name of a manual page (with secondary entries, if they exist) and an abstract of that page. Each line of the "Permuted Index" represents a permutation (or sorting) of a line from the "Table of Contents" into three columns. The lines are arranged so that a keyword or phrase begins the middle column. Use the "Permuted Index" by searching this middle column for a topic or command. When the desired entry has been found, the right column of that line lists the name of the manual page on which information corresponding to that keyword can be found. The left column contains the remainder of the permutation that began in the middle column.

Table of Contents

Table of Contents

1. Commands

7. Special Interfaces

Permuted Index

Permuted Index

Commands (1M)

NAME

intro - introduction to system administrator's commands

DESCRIPTION

This section describes system administrative and maintenance commands. A portion of the commands are standard UNIX™ System V commands that have been modified under CLIX. The remainder are CLIX-specific commands.

COMMAND SYNTAX

Unless otherwise noted, commands described in this section accept options and other arguments according to the following syntax:

name [-*option* ...] [*cmdarg* ...]

where:

[ ]	Surround an *option* or *cmdarg* that is not required.
{ }	Surround a set of *options* or *cmdargs*, one of which must be chosen.
...	Indicates multiple occurrences of the *option* or *cmdarg*.
name	The name of an executable file.
option	(Always preceded by a "-".) *noargletter* ... or *argletter optarg* [,...]
noargletter	A single letter representing an option without an option-argument. Note that more than one *noargletter* option can be grouped after one "-".
argletter	A single letter representing an option requiring an option-argument.
optarg	An option-argument (character string) satisfying a preceding *argletter*. Note that groups of *optargs* following an *argletter* must be separated by commas, or by white space and quoted.
cmdarg	Path name (or other command argument) *not* beginning with "-", or "-" by itself indicating the standard input.

SEE ALSO

getopt(1) in the *UNIX System V User's Reference Manual.*
getopt(3C) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

Upon termination, each command returns two bytes of status, one supplied by the system and giving the cause for termination, and (in the case of "normal" termination) one supplied by the program (see *wait*(2) and *exit*(2)). The former byte is 0 for normal termination; the latter is customarily 0 for successful execution and nonzero to indicate problems such as erroneous parameters, bad or inaccessible data, or another inability to cope with the immediate task. It is called variously "exit code", "exit status", or "return code", and is described only where special conventions are involved.

NAME

acct: acctdisk, acctdusg, accton, acctwtmp - overview of accounting and miscellaneous accounting commands

SYNOPSIS

/usr/lib/acct/acctdisk

/usr/lib/acct/acctdusg [**-u** *file*] [**-p** *passwdfile*]

/usr/lib/acct/accton [*file*]

/usr/lib/acct/acctwtmp *reason*

DESCRIPTION

Accounting software consists of executables and shell procedures that can be used to build accounting systems. *acctsh*(1M) describes the system-supplied set of shell procedures built on top of the executables.

Connect-time accounting (when a user logs in and out) is handled by various programs. These programs write records to **/etc/utmp** and **/etc/wtmp**. Both files are described in *utmp*(4). **/etc/utmp** contains the current logins. **/etc/wtmp** contains a record of logins since the last time the file was created. The programs described in *acctcon*(1M) convert the information in **/etc/wtmp** into session and charging records, which are then summarized by *acctmerg*(1M).

The CLIX system kernel performs process accounting. When a process terminates, a record is written to a file (normally **/usr/adm/pacct**). The programs in *acctprc*(1M) summarize the data in this file. *acctcms*(1M) summarizes command usage. Current process data may be examined using *acctcom*(1).

acctmerg(1M) can merge and summarize process accounting, connect-time accounting, and any accounting records in the format described in *acct*(4). *prtacct*(1M) formats any or all accounting records (see *acctsh*(1M)).

Disk accounting information is kept within the file system. It can be obtained using either *acctdusg* or *diskusg*(1M). *acctdusg* computes disk usage for each login directory. It reads standard input (usually from **find -print**) and computes disk resource consumption (including indirect blocks) for each login. If **-u** is given, records consisting of file names for which *acctdusg* does not charge are placed in *file* (a potential source for finding users trying to avoid disk charges). If **-p** is given, *passwdfile* is the name of the password file. *diskusg*(1M) computes disk usage from the file system i-nodes. Both programs output lines containing user ID, login name, and the number of disk blocks used. *acctdisk* processes this output to convert it to the *acct* structure format (see *acct*(4)). The output can then be merged with other accounting data. *dodisk*(1M) is a shell procedure usually used for disk accounting (see *acctsh*(1M)).

accton will attempt to turn process accounting on if *file* is given. Otherwise *accton* turns process accounting off. *File* must be the name of an existing file (usually **/usr/adm/pacct**) to which process accounting records are appended (see *acct(2)* and *acct*(4)). *accton* is normally accessed from the shell procedure, *turnacct*(1M) (see *acctsh*(1M)).

acctwtmp writes a *utmp*(4) record to standard output. The record contains the current time and a string of characters that describe *reason*. A record type of ACCOUNTING is assigned (see *utmp*(4)). *Reason* must be a string of 11 or fewer characters, numbers, **$**, or spaces.

EXAMPLES

The following commands may be used during reboot and shutdown procedures, respectively:

```
acctwtmp uname >> /etc/wtmp
acctwtmp "file save" >> /etc/wtmp
```

FILES

/etc/passwd	used for login name to user ID conversions
/usr/lib/acct	holds all accounting commands in section (1M)
/usr/adm/pacct	current process accounting file
/etc/wtmp	login/logout history file

SEE ALSO

acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), diskusg(1M), fwtmp(1M), runacct(1M).
acctcom(1) in the *CLIX Programmer's & User's Reference Manual*.
acct(2), acct(4), utmp(4) in the *UNIX System V Programmer's Reference Manual*.

NAME

acctcms - command summary from process accounting records

SYNOPSIS

/usr/lib/acct/acctcms [-cjnst] [-a [-po]] *file* ...

DESCRIPTION

acctcms reads one or more *files* in the format described in *acct*(4). It counts all occurrences of processes that executed identically named commands, sorts them, and writes them to standard output using an internal summary format.

The following options are available:

-a Print output in ASCII rather than in the internal summary format. The output includes command name, number of times executed, total kcore-minutes, total CPU minutes, total real minutes, mean size (in 1024 byte blocks), mean CPU minutes per invocation, hog factor, characters transferred, and blocks read and written, as in *acctcom*(1). Output is normally sorted by total kcore-minutes.

-c Sort by total CPU time rather than total kcore-minutes.

-j Combine all commands invoked only once under *****other**.

-n Sort by number of command invocations.

-s Put any file names specified after this option in internal summary format.

-t Process all records as total accounting records. The default internal summary format splits each field into prime-time and non-prime-time usage. This option combines the prime-time and non-prime-time usage into a single field that contains the total of both. It also provides upward compatibility with old-style (UNIX System V) *acctcms* internal summary format records.

The following options may be used only with the **-a** option.

-p Output a prime-time only command summary.

-o Output a non-prime-time only (offshift) command summary.

When **-p** and **-o** are used together, a combination prime-time and non-prime-time report is produced. Number of times executed, CPU minutes, and real minutes will be summarized into prime-time and non-prime-time usage. All other output summaries will be a combination of the prime-time and non-print-time usage.

EXAMPLES

A typical sequence for performing daily command accounting and for maintaining a running total is as follows:

```
acctcms file ... >today
cp total previoustotal
acctcms -s today previoustotal >total
```

acctcms -a -s today

SEE ALSO

acct(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M).
acctcom(1) in the *CLIX Programmer's & User's Reference Manual.*
acct(2), acct(4), utmp(4) in the *UNIX System V Programmer's Reference Manual.*

CAVEATS

Unpredictable output results if **-t** is used with new-style internal summary format files or if it is not used with old-style internal summary format files.

NAME

acctcon: acctcon1, acctcon2 - connect-time accounting

SYNOPSIS

/usr/lib/acct/acctcon1 [-pt] [-l *sumfile* **] [-o** *record-file* **]**

/usr/lib/acct/acctcon2

DESCRIPTION

acctcon1 converts a sequence of login/logout records read from standard input (normally redirected from **/etc/wtmp**) to a more usable, intermediate form. Its output, in ASCII, includes device, user ID, login name, prime-time usage (seconds), non-prime-time usage (seconds), session starting time (numeric), and starting date and time.

The following options are available:

-p Print input only, showing line name, login name, and time (in both numeric and date/time formats).

-t Instead of letting *acctcon1* use the current time, cause it to use the last time found in its input as the ending time for each session still in progress. *acctcon1* maintains a list of lines on which users are logged in. When it reaches the end of its input, it emits a session record for each line that still appears to be active. *acctcon1* assumes its input is a current file and therefore uses the current time. Thus, it assures reasonable and repeatable numbers for noncurrent files.

-l *sumfile* Create *sumfile* to contain a summary of line usage showing name of line, number of minutes used, percentage of total elapsed time used, number of sessions charged, number of logins, and number of logouts. *Sumfile* helps track line usage, identify bad lines, and find software and hardware oddities. Hangup, termination of *login*(1), and termination of the login shell each generate logout records so that the number of logouts is often three to four times greater than the number of sessions (see *init*(1M) and *utmp*(4)).

-o *record-file* Fill *record-file* with an overall record for the accounting period, giving starting time, ending time, number of reboots, and number of date changes.

acctcon2 converts the records output from *acctcon1* into the *tacct* structure format (see *acct*(4)). *acctcon2* reads from standard in and writes to standard out.

EXAMPLES

These commands are typically used by *runacct*(1M) as shown below. The file "ctmp" is created only for *acctprc*(1M):

```
acctcon1 -t -l lineuse -o reboots <wtmp | sort +1n -2 >ctmp
acctcon2 <ctmp | acctmerg >ctacct
```

FILES

/etc/wtmp login/logout summary

SEE ALSO

acct(1M), acctcms(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M).
init(1M) in the *UNIX System V System Administrator's Reference Manual.*
login(1), acct(2), acct(4), utmp(4) in the *UNIX System V Programmer's Reference Manual.*

BUGS

The line usage report is confused by date changes. Use *wtmpfix*(1M) to correct this situation (see *fwtmp*(1M)).

NAME

acctmerg - merge or add total accounting files

SYNOPSIS

/usr/lib/acct/acctmerg [*option* ...] [*file* ...]

DESCRIPTION

acctmerg reads from standard input and up to nine additional files. All of the files are in the *tacct* structure format (see *acct*(4)) or an ASCII version thereof. It merges the input by adding records whose keys (normally user ID and name) are identical, and sorts the input on those keys. Options are as follows:

-a Produce output in ASCII version of the *tacct* structure.

-i Input files are in ASCII version of the *tacct* structure.

-p Print input with no processing.

-t Produce a single record that totals all input.

-u Summarize by user ID, rather than user ID and name.

-v Produce output in verbose ASCII format, with more precise notation for floating-point numbers.

EXAMPLES

The following sequence is useful for repairing any file kept in this format:

```
acctmerg -v <file1 >file2
```

Edit "file2" as desired.

```
acctmerg -i <file2 >file1
```

SEE ALSO

acct(1M), acctcms(1M), acctcon(1M), acctprc(1M), acctsh(1M), fwtmp(1M), runacct(1M).
acctcom(1) in the *CLIX Programmer's & User's Reference Manual.*
acct(2), acct(4), utmp(4) in the *UNIX System V Programmer's Reference Manual.*

NAME

acctprc - process accounting

SYNOPSIS

/usr/lib/acct/acctprc [*ctmp*]

DESCRIPTION

acctprc reads input in *acct*(4) form, inserting login names corresponding to the user ID. The data is then summarized by user ID and name, and written to standard output in the *tacct* structure format (see *acct*(4)). If *ctmp* is given, it should contain a list of login sessions, in the form described in *acctcon*(1M), sorted by user ID and login name. If *ctmp* is not given, *acctprc* obtains login names from **/etc/passwd**. The information in *ctmp* distinguishes different login names that share the same user ID.

EXAMPLES

This command is typically used as follows:

```
acctprc ctmp </usr/adm/pacct >ptacct
```

FILES

/etc/passwd user account information

SEE ALSO

acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctsh(1M), fwtmp(1M), runacct(1M).
acctcom(1) in the *UNIX System V User's Reference Manual.*
acct(2), acct(4), utmp(4) in the *UNIX System V Programmer's Reference Manual.*
cron(1M) in the *UNIX System V System Administrator's Reference Manual.*

CAVEATS

Although distinguishing login names that share user IDs interactively is possible, more precise conversion is possible by using *acctwtmp*(1M).

NAME

acctsh: chargefee, ckpacct, dodisk, lastlogin, monacct, nulladm, prctmp, prdaily, prtacct, shutacct, startup, turnacct - shell procedures for accounting

SYNOPSIS

/usr/lib/acct/chargefee *login-name number*

/usr/lib/acct/ckpacct [*blocks*]

/usr/lib/acct/dodisk [**-o**] [*file-system* ...]

/usr/lib/acct/lastlogin

/usr/lib/acct/monacct *number*

/usr/lib/acct/nulladm *file*

/usr/lib/acct/prctmp

/usr/lib/acct/prdaily [**-l**] [**-c**] [*mmdd*]

/usr/lib/acct/prtacct *file* [*heading*]

/usr/lib/acct/shutacct [*reason*]

/usr/lib/acct/startup

/usr/lib/acct/turnacct {**on** | **off** | **switch**}

DESCRIPTION

chargefee charges a *number* of units to *login-name*. A record is written to **/usr/adm/fee** to be merged with other accounting records at the time specified with *runacct*(1M). (In most cases, this time occurs during the night.)

ckpacct checks the size of **/usr/adm/pacct** and should be invoked by *cron*(1M). If the size exceeds *blocks* (1000 by default), *turnacct* will be invoked with the **switch** argument. If the number of free disk blocks in the **/usr** file system falls below 500, *ckpacct* will automatically turn off the collection of process accounting records with the **off** argument to *turnacct*. When 500 blocks or more are restored, accounting will be activated again.

dodisk performs the disk accounting functions and should be invoked by *cron*(1M). By default, it will perform disk accounting on the special files in **/etc/fstab**. If the **-o** flag is specified, *dodisk* will run disk accounting based on login directory, which will be slower. *File-system* specifies one or more file system names where disk accounting will occur. If *file-systems* are specified, disk accounting will be performed on these file systems only. If the **-o** flag is specified, *file-systems* should be directories on which file systems are mounted. If **-o** is omitted, *file-systems* should be the special file names of mountable file systems.

lastlogin updates **/usr/adm/acct/sum/loginlog**, which shows the last date on which each person logged in. It is invoked by *runacct*(1M).

monacct should be invoked once each month or each accounting period. *Number* indicates which month or period it is invoked. If *number* is not given, it defaults to the current month (01-12). This default is useful if *monacct* will execute with *cron*(1M) on the first day of each month. *monacct* creates summary files in **/usr/adm/acct/fiscal** and restarts summary files in **/usr/adm/acct/sum.**

nulladm creates *file* with mode 664 and ensures that owner and group are the adm account. It is called by various accounting shell procedures.

prctmp prints the session record file (usually **/usr/adm/acct/nite/ctmp** created by *acctcon*(1M)).

prdaily formats a report of the previous day's accounting data; it is invoked by *runacct*(1M). The report resides in **/usr/adm/acct/sum/rprt***mmdd*, where *mmdd* is the month and day of the report. The user can print the current daily accounting reports by executing *prdaily*. Previous accounting reports can be printed with the *mmdd* option specifying the exact report date desired. The **-l** flag prints a report of exceptional usage by login ID for the specified date. *monacct* deletes previous daily report. The **-c** flag prints a report of exceptional resource usage by command and may be used on the current day's accounting data only.

prtacct formats and prints any total accounting file in *tacct* structure format.

shutacct is invoked during a system shutdown to turn process accounting off and append a *reason* record to **/etc/wtmp.**

startup turns accounting on when the system acquires a multiuser state.

turnacct is an interface to *accton* to turn process accounting **on** or **off** (see *acct*(1M)). The **switch** argument turns accounting off, moves the current **/usr/adm/pacct** to the next free name in **/usr/adm/pacct***incr* (where *incr* is a number starting with 1 and incrementing by 1 for each additional **pacct***incr* file), and then turns accounting back on again. *turnacct* is called by *ckpacct* and thus can be invoked by *cron*(1M) and therefore maintain **pacct***incr* at a reasonable size. *turnacct* starts and stops process accounting using *init*(1M) and *shutdown*(1M), respectively.

FILES

/usr/adm/fee	accumulator for fees
/usr/adm/pacct	current file for per-process accounting
/usr/adm/pacct*	used during daily accounting execution
/etc/wtmp	login/logout summary
/usr/lib/acct/ptelus.awk	limits for exceptional usage by login ID
/usr/lib/acct/ptecms.awk	limits for exceptional usage by command name
/usr/adm/acct/nite	working directory
/usr/lib/acct	holds all accounting commands in section (1M)
/usr/adm/acct/sum	summary directory that should be saved

SEE ALSO

acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), diskusg(1M), fwtmp(1M), runacct(1M).
acctcom(1) in the *CLIX Programmer's & User's Reference Manual.*

cron(1M) in the *UNIX System V System Administrator's Reference Manual.*
acct(2), acct(4), utmp(4) in the *UNIX System V Programmer's Reference Manual.*

NAME

arp - address resolution display and control

SYNOPSIS

arp *host-name*
arp -a
arp -d *host-name*
arp -s *host-name ether-addr* [**temp**] [**pub**]
arp -f *file-name*
arp -t *host-name*

DESCRIPTION

arp displays and modifies the Internet-to-Ethernet address translation tables used by the Address Resolution Protocol (ARP).

With no flags, the program displays the current ARP entry for *host-name*. The host may be specified by name or by number, using Internet dot notation.

arp accepts the following options:

-a Display all of the current ARP entries.

-d *host-name*
Delete the ARP entry for host *host-name*. Super-user privilege is required for this option.

-s *host-name ether-addr* [**temp**] [**pub**]
Create an ARP entry for host *host-name* with the Ethernet address *ether-addr*. The Ethernet address is given as six hexadecimal bytes separated by dashes. The entry will be permanently added to the ARP tables unless the **temp** option is used. The node will respond to any ARP request seen on the network for host *host-name* if the **pub** option is used. Super-user privilege is required for the **-s** option.

-f *file-name*
Read from *file-name* and set multiple entries in the *arp* tables. Entries in *file-name* have the following format:

host-name ether-addr [**temp**] [**pub**]

Super-user privilege is required for this option.

-t *host-name*
Instead of using the local table to get the translation, transmit a request on the network to obtain the proper translation.

SEE ALSO

inet(7B), arp(7B).

NAME

auditd - file system event logging and report generation daemon

SYNOPSIS

/etc/auditd [-rdvh] [-c *file*] [-u *size*]

DESCRIPTION

auditd is a multipurpose audit daemon and report generator for selected file system events (see *audit*(7S)). Invoking *auditd* without any arguments causes auditing to be initialized on the system and an audit report to be sent to standard out. The audit records produced are English text; further decryption is not necessary.

auditd supports the following options:

-h Print help.

-r Dump audit records in raw format. This is useful if further processing is desired on the audit data.

-v Enable verbose mode.

-d Run *auditd* process as a daemon in the background.

-c *file* Specify a file other than **/dev/audit** to read audit records from. This is useful for interpreting previously generated raw audit records. This option will not work with **-d**.

-u *size* Set the maximum size of all files created by the audit daemon. The default is 100,000 disk blocks. Note that if the *size* is exceeded, warnings are not given and the file is truncated to *size*.

EXAMPLES

If the report needs to be saved for review later, execute the following command:

```
/etc/auditd -r -d > /usr/adm/adt/adt.log
```

This will write the data in raw form (see *adt*(7S)) to the file "/usr/adm/adt/adt.log". To review the audit data saved previously, execute the following command:

```
/etc/auditd -c /usr/adm/adt/adt.log
```

FILES

/dev/audit audit device

SEE ALSO

adt(7S).

NAME

bootcp - boot the Communication Processor

SYNOPSIS

/etc/bootcp [-v] [*file*]

DESCRIPTION

bootcp downloads the Communication Processor (CP) with the code specified by *file* or the default code **/boot/cp.ima**. The code is checked for validity before being loaded.

If *bootcp* cannot find *file* or the default file, the user will be prompted to specify an alternate file. A <RETURN> will terminate *bootcp* and no code will be loaded.

The -v option causes diagnostic messages to be output during the download process.

FILES

/boot/cp.ima default code file

DIAGNOSTICS

If any error occurs during the download process, a message is printed and *bootcp* exits with a nonzero status.

NAME

bootinfo - display header information of a bootable image

SYNOPSIS

bootinfo *file* ...

DESCRIPTION

bootinfo displays the header information of an Intergraph® boot image. It lists the checksum, size in 512-byte blocks, and the date of the image.

FILES

/dev/rdsk/s0u0p8.7

SEE ALSO

bootheader(4) in the *CLIX Programmer's & User's Reference Manual.*

BUGS

/f2bootinfo/fP does not check the magic number of the boot file specified. Therefore, it is possible to get seemingly valid data from any file.

NAME

btp_listener, btp - BTP listener and configuration utility

SYNOPSIS

/usr/ip32/xnsvtp/btp_listener [**-v**] [**-l** *logfile*] [**-w** *statfile*]

btp [**-b** *address* | **-u** *address*] [**-d** *domain*] [**-o** *organization*] [**-p** *globalpasswd*] [**-s**]

DESCRIPTION

btp_listener, the Bridge Transaction Protocol (BTP) listener, allows CS/200 and CS/210 Terminal Servers to boot from CLIX hosts. *btp_listener* allows the user to set and store all communications server port configurations, global parameters, passwords, and macros. It also enables communications servers to resolve addresses from node names using a CLIX host's Intergraph Clearinghouse.

The following options are available to *btp_listener*:

-v Display the version. If no other options are specified, the process will exit. The use of this option will not affect a running *btp_listener*.

-l *logfile* Start *btp_listener* with logging to *logfile*.

-w *statfile* Start *btp_listener* with statistical data written to the *statfile*.

btp is the configuration utility for *btp_listener*. The following options are available:

-b *address* Bind the CS/200 or CS/210 Terminal Server at *address* to the local host. *address* consists of the last six hexadecimal digits of the terminal server Ethernet address.

-d *domain* Set the default domain to be the string *domain*.

-o *organization* Set the default organization description to the string *organization*.

-p *globalpasswd* Set the global password to the string *globalpasswd*.

-s Display the default parameter values.

-u *address* Unbind the terminal server at *address* from the local host. *address* consists of the last six hexadecimal digits of the terminal server Ethernet address.

SEE ALSO

XNS/VTP Administrator's Guide.

NAME

crr_listener - listener for XNS/VTP remote login requests

SYNOPSIS

/usr/ip32/xnsvtp/crr_listener [-v] [-l *logfile*]

DESCRIPTION

crr_listener monitors Xerox Network Services/Virtual Terminal Protocol (XNS/VTP) connection requests that are generated by 3COM Communications Servers and *visit*(1) connection requests that specify the **-p vtp** option. Whenever a request is received, *crr_listener* invokes a server that completes the connection and returns to listening for other connection requests. The following options are available:

-v Display the version. If no other options are specified, the process will exit and not start the *crr_listener*. This option will not affect a *crr_listener* that is currently executing.

-l *logfile* Log informational and error messages to the *logfile*.

SEE ALSO

visit(1) in the *CLIX Programmer's & User's Reference Manual*.
XNS/VTP Administrator's Guide.

NAME

cumaild - DNP cumail(1) server

SYNOPSIS

cumaild

DESCRIPTION

cumaild, the Digital Network Protocol (DNP) mail server, processes all incoming mail from the DECnet network and uses the local mail program to deliver mail messages. The mail server is invoked through *netserver*(1M).

FILES

/usr/lib/servers.reg	registered server list

SEE ALSO

netserver(1M).
cumail(1) in the *CLIX Programmer's & User's Reference Manual.*

NAME

dates - display information for Intergraph products

SYNOPSIS

dates

DESCRIPTION

dates extracts and displays the product name, number, date, version, and title from **/usr/ip32/ingrconfig**. This file is maintained by the *newprod*(1M) utility.

dates also compares the date found in **/usr/ip32/ingrconfig** with the dates of the product's **fixes.com** file. If the dates do not match, the **fixes.com** date appears in parentheses at the end of the product line.

FILES

/usr/ip32/ingrconfig	delivery configuration file
/usr/ip32/*/fixes.com	product fixes file

SEE ALSO

newprod(1M).
fixes.com(4) in the *CLIX Programmer's & User's Reference Manual.*

NAME

df - report number of free disk blocks and i-nodes

SYNOPSIS

df [**-lt**] [**-f**] [*file-system* | *directory* | *mounted-resource*]

DESCRIPTION

df prints out the number of free blocks and free i-nodes in mounted file systems, directories, or mounted resources by examining the counts kept in the super-blocks.

File-system may be specified either by a device name (e.g., **/dev/dsk/s0u0p7.3**) or by a mount point directory name (e.g., **/usr**).

Directory can be a directory name. The report presents information for the device that contains the directory.

Mounted-resource can be a remote resource name. The report presents information for the remote device that contains the resource.

The device or mounted file system can be an *fs*(4) type file system, in which case *df* prints the information. If the device is unmounted and is not an *fs*(4) type file system, then *df* executes the *fstyp*(1M) program, forms a path name from the output, and executes the program so formed. For example, if *fstyp* reports the file system to be of type "FFS", *df* will execute *ffsdf*(1M).

If no arguments are used, the free space on all locally and remotely mounted file systems is printed.

The *df* command uses the following options:

-l Only reports on local file systems.

-t Causes the figures for total allocated blocks and i-nodes to be reported as well as the free blocks and i-nodes.

-f an actual count of the blocks in the free list is made, rather than taking the figure from the super-block (free i-nodes are not reported). This option will not print any information about mounted remote resources.

FILES

/dev/dsk/*
/etc/mnttab

SEE ALSO

mount(1M).
fs(4), mnttab(4) in the *CLIX Programmer's & User's Reference Manual.*
fstyp(1M) in the *UNIX System V System Administrator's Reference Manual.*

NOTES

If multiple remote resources are listed that reside on the same file system on a remote machine, each listing after the first one will be marked with an asterisk.

NAME

diskusg - generate disk accounting data by user ID

SYNOPSIS

diskusg [*options* ...] [*file* ...]

DESCRIPTION

diskusg generates intermediate disk accounting information from data in *files*, or standard input if *files* is omitted. *diskusg* directs output to standard output with one line per user displaying *uid*, *login*, and *#blocks*, respectively. The format of each line is explained as follows:

uid	the numerical user ID of the user
login	the login name of the user
#blocks	the total number of disk blocks allocated to this user

diskusg normally reads only the i-nodes of file systems for disk accounting. In this case, *files* are the special file names of these devices.

diskusg recognizes the following options:

-s The input data is already in *diskusg* output format. *diskusg* combines all lines for a single user into a single line.

-v Verbose. Print a list on standard error of all files that are not charged to a user.

-i *fnmlist* Ignore the data on file systems whose names are in *fnmlist*. *Fnmlist* is a list of file system names separated by commas or enclosed within quotation marks. *diskusg* compares each name in this list with the file system name stored in the volume ID (see *labelit*(1M)).

-p *file* Use *file* as the name of the password file to generate login names. **/etc/passwd** is used by default.

-u *file* Write records for files that are not charged to a user to *file*. Records consist of the special file name, the i-node number, and the user ID.

The output of *diskusg* is normally the input to *acctdisk*(1M) (see *acct*(1M)) that generates total accounting records that can be merged with other accounting records. *diskusg* is normally run by *dodisk*(1M) (see *acctsh*(1M)).

EXAMPLES

The following will generate daily disk accounting information for root on "/dev/dsk/c1d0s0":

```
diskusg /dev/dsk/c1d0s0 | acctdisk >disktacct
```

FILES

/etc/passwd used for user ID to login name conversions

SEE ALSO

acct(1M), acctsh(1M).
acct(4) in the *UNIX System V Programmer's Reference Manual.*

NAME

dodini - initializes/enables the DoD suite of protocols

SYNOPSIS

/etc/dodini -a [*Internet-addr*] [**-r** [*file-name*]]
/etc/dodini -r [*file-name*]

DESCRIPTION

The **-a** option is used to specify an address and the **-r** option is used to specify a static routing table.

Internet-addr can be specified in a "dot" format (such as **192.9.200.1**) or as a node name (such as **foobar**). If the **-a** option is used with no address specified, the local address is used.

File-name is the path name of an ASCII file containing the static routing table. This table is used when non-Intergraph Internet protocol gateways reside on the local network. Static routing tables are only needed if non-Intergraph equipment is on the network.

If the -r option is used with no file name specified, the file **/etc/iprtab** will be used.

The routing table consists of one entry per line in the following format:

NETWORK: *network* **GATEWAY:** *gateway address* ;*comments* ...

Network is an Internet protocol network number in dotted decimal notation and *gateway address* is the address of the local gateway to use. Any *comments* are ignored by *dodini*.

NAME

errconfig - error log daemon manager

SYNOPSIS

errconfig [-**hgrnc**] [-**s***node*] [-**f***file*] [-**m***size*] [-**o***types*]

DESCRIPTION

errconfig, the error log daemon manager, sends requests to the error log daemon, *errord*(1M), to display or change the configuration of *errord*(1M) while the daemon is running. The available options are as follows:

-h Display a help screen.

-g Display the current configuration of *errord*(1M). This is the default if no options are specified.

-m *size* Specify the maximum allowable number of blocks in the error log file. When the error log file reaches this size, it is renamed **/usr/adm/olderrorlog** and a new error log file is created. The default is 500 blocks.

-s *node* Send error messages to system *node*. If this option is specified, *errord*(1M) will post errors to the local system and to system *node*. For *node* to receive these errors, the *errord*(1M) running on *node* must be executed with the **-r** option.

-r Receive error messages from other systems and put them in **/usr/tmp/errlog**. Unless this option is specified, the local system will not receive errors sent from remote error daemons executed with the **-s** option.

-n Do not send or receive error messages from remote systems. The **-n** option is the default.

-c Write the current configuration of *errord*(1M) to **/usr/adm/errord.rc**.

-f *file* Specify the error log file to use. The default is **/usr/adm/errlog**.

-o *types* Exclude the error types specified by *types* from the error log file. Valid error types are **device**, **user**, **panic**, **memory**, **slave**, **disk**, **tape**, **floppy**, **asycn**, **scan**, **parallel**, **digitizer**, **timeout**, **security**, **stray**, **optic**, **soft**, **retry**, and **hard**. If multiple *types* are specified, they must be separated by commas and/or spaces; if spaces are used, the entire *types* string must be enclosed in quotation marks.

EXAMPLES

The following command displays the current configuration of *errord*(1M):

```
errconfig
```

The following command causes *errord*(1M) to send error messages to node "bike" but not to log or send **disk** and **memory** errors. The changes to the configuration are saved in **/usr/adm/errord.rc** after they are made.

errconfig -s bike -o disk,memory -c

FILES

/usr/adm/errord.rc error daemon configuration file

SEE ALSO

errord(1M).
errord.rc(4) in the *CLIX Programmer's & User's Reference Manual.*

NOTES

errconfig either examines (**-g**) or changes (all other options) the configuration of *errord*(1M). The **-g** option should not be specified on the command line with the change options.

NAME

errord - error logging daemon

SYNOPSIS

/usr/lib/errord

DESCRIPTION

errord, the error log daemon invoked at boot time, posts system errors to the error log file. Upon starting, the daemon checks for the existence of the configuration file, **/usr/adm/errord.rc**. If **/usr/adm/errord.rc** does not exist, *errord* will use the default configuration, which is to log errors to **/usr/adm/errlog**, limit the size of the log file to 500 blocks, and log all reported errors.

The kernel posts errors to **/dev/errorlog**. When an error is posted, *errord* reads **/dev/errorlog** and appends the message to the log file. If the log file does not exist, *errord* will create it. If the log file becomes larger than 500 blocks, the daemon will rename it **/usr/adm/olderrlog** and create a new error log file.

errord can send errors directly to the error report generator, *errors*(1). In addition, it can send error messages to or receive them from other systems on the network and selectively log error types. See *errord.rc*(4) and *errconfig*(1M) for more details on customizing the configuration of the daemon.

FILES

/dev/errorlog	system error log device
/usr/adm/errlog	default system error log file
/usr/adm/olderrlog	old system error log file
/usr/adm/errord.rc	error daemon configuration file

SEE ALSO

errconfig(1M).
errord.rc(4), errors(1) in the *CLIX Programmer's & User's Reference Manual.*

WARNINGS

Transmission Control Protocol/Internet Protocol (TCP/IP) must be running for the daemon to send or receive errors from other systems.

/dev/errorlog must exist for *errord* to execute.

NAME

fal - DNP file access server

SYNOPSIS

fal

DESCRIPTION

fal, the Digital Network Protocol (DNP) File Access Listener, is the server that enables users on remote hosts to access files on the local host. Specifically, *fal* accesses the local file system to provide the remote user network file access functions such as file transfer and directory listings. *fal* uses the Data Access Protocol (DAP) to communicate with processes on other nodes.

SEE ALSO

netserver(1M).
netcp(1), netlpr(1), netls(1), netmv(1), netrm(1), netex(1) in the *CLIX Programmer's & User's Reference Manual.*

NAME

ffsdf - report free disk blocks and i-nodes on unmounted fast file system

SYNOPSIS

/etc/ffsdf [-lt] [-f] *special*

DESCRIPTION

ffsdf reports file system statistics on unmounted Fast File System (FFS) *special* devices by examining counts kept in the super-blocks.

FILES

/dev/dsk/*

SEE ALSO

df(1M).

NOTES

ffsdf is a special case of the *df*(1M) command and is normally executed by *df*(1M). *df*(1M) is preferred to *ffsdf* for examining file systems.

NAME

ffsfsck - fast file system consistency check and interactive repair

SYNOPSIS

/etc/ffsfsck [**-B** *block#*] [**-y**] [**-n**] [**-b**] [*file-system* ...]

DESCRIPTION

ffsfsck audits and interactively repairs inconsistent conditions for file systems of type *ffsfs*(4). If the file system is inconsistent, the operator is prompted for confirmation before each correction is attempted. Some of the corrective actions will result in some data loss. The amount and severity of lost data may be determined from the diagnostic output. The default action for each consistency correction is to wait for the operator to respond "yes" or "no". If the operator does not have write permission on the file system, *ffsfsck* will default to a **-n** action.

ffsfsck is called by *fsck*(1M) for file systems of type *ffsfs*(4). After correcting a file system, *ffsfsck* will print the number of files on that file system, the number of used and free blocks, and the percentage of fragmentation.

If sent a QUIT signal, *ffsfsck* will finish the file system checks and then exit with a nonzero return status.

The following options are available:

-B *block#* Use *block#* as the super-block for the file system. Block 32 is always an alternate super-block.

-y Assume a "yes" response to all questions asked by *ffsfsck*; this should be used with caution.

-n Assume a "no" response to all questions asked by *ffsfsck*; do not open the file system for writing.

-b Reboot. If the file system being checked is the root file system and modifications have been made, either remount the root file system or reboot the system depending on the extent of the modifications. Remount only if there was minor damage.

The following inconsistencies are checked:

1. Blocks claimed by more than one i-node or the free list.
2. Blocks claimed by an i-node or the free list outside the range of the file system.
3. Incorrect link counts.
4. Directory size not in proper format.
5. Bad i-node format.
6. Blocks not accounted for.
7. Directory checks including file pointing to unallocated i-node and i-node number out of range.
8. More blocks for i-nodes than the file system has.
9. Bad free block list format.

10. Total free block and/or free i-node count incorrect.

Orphaned files and directories (allocated but unreferenced) are (with the operator's concurrence) reconnected by placing them in the **lost+found** directory. The name assigned is the i-node number. If the file system's **lost+found** directory does not exist, it is created. If space is insufficient, its size is increased.

SEE ALSO

newfs(1M), ffsmkfs(1M), fsck(1M).

DIAGNOSTICS

The diagnostics produced by *ffsfsck* are explained in the "FFS Check Tutorial" in the *CLIX System Guide*.

NAME

ffsfsstat - report file system status

SYNOPSIS

/etc/ffsfsstat *special-file*

DESCRIPTION

ffsfsstat reports the status of the file system on *special-file*, which must be type *ffsfs*(4). *ffsfsstat* can be called from the *fsstat*(1M) command.

During startup, this command determines if the file system needs to be checked before it is mounted. *ffsfsstat* succeeds if the file system is unmounted and appears to be okay. For the root file system, it succeeds if the file system is active and not marked as bad.

SEE ALSO

df(1M), fsstat(1M).
ffsfs(4) in the *CLIX Programmer's & User's Reference Manual.*

DIAGNOSTICS

The command has the following exit codes:

0 The file system is not mounted and appears okay (except for root where 0 means mounted and okay).

1 The file system is not mounted and needs to be checked.

2 The file system is mounted.

3 The command failed.

NAME

ffsmkfs - construct a file system

SYNOPSIS

/etc/ffsmkfs [-N] *special size* [*nsect* [*ntrack* [*blksize* [*fragsize* [*ncpg* [*minfree* [*rps* [*nbpi* [**s** | **t**]]]]]]]]]

DESCRIPTION

ffsmkfs constructs a file system by writing *special* on the special file unless the **-N** option has been specified. The numeric *size* specifies the number of sectors in the file system. *ffsmkfs* builds a file system with a root directory and a **lost+found** directory (see *ffsfsck*(1M)). The number of i-nodes is calculated as a function of the file system size.

The optional arguments allow fine tune control over the parameters of the file system. *Nsect* specifies the number of sectors per track on the disk. *Ntrack* specifies the number of tracks per cylinder on the disk. *Blksize* gives the primary block size for files on the file system. It must be a power of two, currently selected from 8192 or 16384. *Fragsize* gives the fragment size for files on the file system. The *fragsize* represents the smallest amount of disk space allocated to a file. It must be a power of two currently selected from the range 512 to 8192. *Ncpg* specifies the number of disk cylinders per cylinder group. This number must be in the range 1 to 32. *Minfree* specifies the minimum percentage of free disk space allowed. Once the file system capacity reaches this threshold, only the super-user is allowed to allocate disk blocks. The default value is 10 percent. If a disk does not revolve at 60 revolutions per second, the *rps* parameter may be specified. If a file system will have more or less than the average number of files, the *nbpi* (number of bytes per i-node) can be specified to increase or decrease the number of i-nodes created. Space or time optimization preference can be specified with either **s** for space or **t** for time. Users with special demands for their file systems are referred to the "FFS Tutorial" for a discussion of the tradeoffs in using different configurations.

SEE ALSO

ffsfsck(1M), newfs(1M).
ffsfs(4) in the *CLIX Programmer's & User's Reference Manual.*
"FFS Tutorial" in the *CLIX System Guide.*

NOTES

File systems are normally created with the *newfs*(1M) command.

BUGS

ffsmkfs does not support bad blocks.

NAME

fmus - server for *fmu*(1)

SYNOPSIS

/usr/ip32/inc/fmus

DESCRIPTION

fmus is the server for *fmu*(1) and can only be invoked by the *xns_listener*(1M).

SEE ALSO

xns_listener(1M).
fmu(1), server.dat(4) in the *CLIX Programmer's & User's Reference Manual.*

NAME

fsck, dfsck - check and repair file systems

SYNOPSIS

/etc/fsck [-**y**] [-**n**] [-**s***X*] [-**S***X*] [-**t** *file*] [-**q**] [-**D**] [-**f**] [-**b**] [*special*]
/etc/dfsck [*options1*] *special1* ... - [*options2*] *special2* ...

DESCRIPTION

Fsck

fsck audits and interactively repairs inconsistent conditions for file systems. If the file system is found to be consistent, the number of files, blocks used, and blocks free are reported. If the file system is inconsistent the user is prompted for concurrence before each correction is attempted. It should be noted that most corrective actions will result in some loss of data. The amount and severity of data loss may be determined from the diagnostic output. The default action for each correction is to wait for the user to respond **yes** or **no**. If the user does not have write permission *fsck* defaults to a -**n** action.

fsck audits an *fs*(4) type file system as the default file system and also acts as a front-end driver for other file system types. If the file system is not the default type, it executes *fstyp*(1M), forms a path name from the output, and executes the audit program so formed. For example, if *fstyp*(1M) reports the file system to be of type "FFS", *fsck* will execute *ffsfsck*(1M).

The following options are accepted by *fsck*.

-**y** Assume a "yes" response to all questions asked by *fsck*.

-**n** Assume a "no" response to all questions asked by *fsck*; do not open the file system for writing.

-**s***X* Ignore the actual free list and (unconditionally) reconstruct a new one by rewriting the super-block of the file system. The file system should be unmounted while this is done; if this is not possible, care should be taken that the system is quiescent and that it is rebooted immediately afterwards. This precaution is necessary so that the old, bad, in-core copy of the super-block will not continue to be used, or written on the file system.

The -**s***X* option allows for creating an optimal free-list organization.

If *X* is not given, the values used when the file system was created are used. The format of *X* is *cylinder-size*:*gap-size*.

-**S***X* Conditionally reconstruct the free list. This option is like -**s***X* above except that the free list is rebuilt only if there were no discrepancies discovered in the file system. Using -**S** will force a "no" response to all questions asked by *fsck*. This option is useful for forcing free list reorganization on uncontaminated file systems.

-t *file* If *fsck* cannot obtain enough memory to keep its tables, it uses a scratch file. If the **-t** option is specified, the file named in the next argument is used as the scratch file, if needed. Without the **-t** flag, *fsck* will prompt the user for the name of the scratch file. The file chosen should not be on the file system being checked, and if it is not a special file or did not already exist, it is removed when *fsck* completes.

-q Quiet *fsck*. Do not print size-check messages. Unreferenced FIFOs will silently be removed. If *fsck* requires it, counts in the super-block will be automatically fixed and the free list salvaged.

-D Directories are checked for bad blocks. Useful after system crashes.

-f Fast check. Check block and sizes and check the free list. The free list will be reconstructed if it is necessary.

-b Reboot. If the file system being checked is the root file system and modifications have been made, then either remount the root file system or reboot the system. A remount is done only if there was minor damage.

If no *special* files are specified, *fsck* will read a list of default file systems from the file **/etc/checklist**.

Inconsistencies checked are as follows:

1. Blocks claimed by more than one i-node or the free list.
2. Blocks claimed by an i-node or the free list outside the range of the file system.
3. Incorrect link counts.
4. Size checks:
   Incorrect number of blocks.
   Directory size not 16-byte aligned.
5. Bad i-node format.
6. Blocks not accounted for anywhere.
7. Directory checks:
   File pointing to unallocated i-node.
   I-node number out of range.
8. Super-block checks:
   More than 65536 i-nodes.
   More blocks for i-nodes than there are in the file system.
9. Bad free block list format.
10. Total free block and/or free i-node count incorrect.

Orphaned files and directories (allocated but unreferenced) are, with the user's concurrence, reconnected by placing them in the file system's **lost+found** directory, if the files are nonempty. The user will be notified if the file or directory is empty or not. Empty files or directories are removed, as long as the **-n** option is not specified. *fsck* will force the reconnection of nonempty directories. The name assigned is the i-node number. The only restriction is that the directory **lost+found** must preexist in the root of the file system being checked and must have empty slots in which entries can be made. This is accomplished by running *mklost+found*(1M) in the root directory of the file system when the file system is created.

Checking the raw device is almost always faster and should be used with everything but the root file system.

Dfsck

dfsck allows two file system checks on two different drives simultaneously. *options1* and *options2* are used to pass options to *fsck* for the two sets of file systems. A - is the separator between the file system groups.

The *dfsck* program permits a user to interact with two *fsck* programs at once. To aid in this, *dfsck* will print the file system name for each message to the user. When answering a question from *dfsck*, the user must prefix the response with a "1" or a "2" (indicating that the answer refers to the first or second file system group).

FILES

/etc/checklist contains default list of file systems to check

SEE ALSO

ffsfsck(1M), mklost+found(1M).
fstyp(1M), ncheck(1M), crash(1M) in the *UNIX System V System Administrator's Reference Manual.*
uadmin(2), checklist(4), fs(4) in the *UNIX System V Programmer's Reference Manual.*

BUGS

I-node numbers for . and .. in each directory are not checked for validity.

NAME

fsstat - report file system status

SYNOPSIS

/etc/fsstat *special*

DESCRIPTION

fsstat reports on the status of the file system on *special*. During startup, this command is used to determine if the file system needs checking before it is mounted. *fsstat* succeeds if the file system is unmounted and appears okay. For the root file system, it succeeds if the file system is active and not marked bad.

If *special* is of type other than *fs*(4) type, *fsstat* determines the file system identifier via the *sysfs*(2) system call, forms a path name from it, and executes the corresponding program. For example, if the file system identifier is "FFS", *fsstat* executes *fsfsstat*(1M).

SEE ALSO

ffsfsstat(1M).
fs(4), sysfs(2) in the *UNIX System V Programmer's Reference Manual.*

DIAGNOSTICS

The command has the following exit codes:

0 The file system is not mounted and appears okay, (except for root where 0 means mounted and okay).

1 The file system is not mounted and needs to be checked.

2 The file system is mounted.

3 The command failed.

NAME

ftpd - Internet FTP server

SYNOPSIS

/usr/ip32/tcpip/ftpd

DESCRIPTION

ftpd is the Internet File Transfer Protocol (FTP) server process. The server uses the Transmission Control Protocol (TCP) and listens at the port specified in the "ftp" service specification (see *services*(4)). The server is normally started by *inetd*(1M).

ftpd currently supports the following ftp requests; case is not distinguished.

Request	Description
ABOR	Abort previous command.
ALLO	Allocate storage (vacuously).
APPE	Append to a file.
CDUP	Change to parent of current working directory.
CWD	Change working directory.
DELE	Delete a file.
HELP	Give help information.
LIST	Give a list of files in a directory ("ls -ls").
MKD	Make a directory.
MODE	Specify data transfer *mode*.
NLST	Give a name list of files in a directory ("ls").
NOOP	Do nothing.
PASS	Specify password.
PASV	Prepare for server-to-server transfer.
PORT	Specify data connection port.
PWD	Print the current working directory.
QUIT	Terminate session.
RETR	Retrieve a file.
RMD	Remove a directory.
RNFR	Specify rename-from file name.
RNTO	Specify rename-to file name.
STOR	Store a file.
STOU	Store a file with a unique name.
STRU	Specify data transfer *structure*.
TYPE	Specify data transfer *type*.
USER	Specify user name.
XCUP	Change to parent of current working directory.
XCWD	Change working directory.
XMKD	Make a directory.
XPWD	Print the current working directory.
XRMD	Remove a directory.

The remaining ftp requests specified in Internet RFC 959 are recognized, but not implemented.

ftpd will timeout an inactive session after 15 minutes.

ftpd will abort an active file transfer only when the ABOR command is preceded by a Telnet "Interrupt Process" (IP) signal and a Telnet "Synch" signal in the command Telnet stream, as described in Internet RFC 959.

ftpd interprets file names according to the "globbing" conventions used by *sh*(1). This allows users to utilize the metacharacters *, ?, [,], {, }, and ~.

ftpd authenticates users according to three rules:

1) The user name must be in the password database, **/etc/passwd**, and not have a null password. In this case a password must be provided by the client before any file operations may be performed. The exception to this is rule 3.

2) The user name must not appear in the file **/etc/ftpusers**.

3) If the user name is "anonymous" or "ftp", one of these accounts must be present in the password file. In these cases, no password is required. *ftpd* takes special measures to restrict the client's access privileges with these accounts.

SEE ALSO

inetd(1M).
ftp(1) in the *CLIX Programmer's & User's Reference Guide.*
sh(1) in the *UNIX System V Programmer's Reference Manual.*

BUGS

The "anonymous" or "ftp" accounts are inherently dangerous and should be avoided when possible.

NAME

fwtmp, wtmpfix - manipulate connect-time accounting records

SYNOPSIS

/usr/lib/acct/fwtmp [-ic]

/usr/lib/acct/wtmpfix [*file* ...]

DESCRIPTION

fwtmp reads from standard input and writes to standard output, converting binary records of the type found in **/etc/wtmp** to formatted ASCII records (see *utmp*(4)).

The following arguments are available:

-i Specify that input is in ASCII form.

-c Specify that output is in binary form.

wtmpfix examines standard input or the named *files*, which should be in *utmp*(4) format, modifies the time and date stamps to make the entries consistent, and writes to standard output. A - can be used instead of *file* to indicate standard input. If the time and date stamps are not modified, *acctcon*(1M) will abort when it encounters certain date-change records.

Each time the date is set, a pair of date change records are written to **/etc/wtmp**. The first record is the old date denoted by the string **old time** placed in the *line* field and the flag OLD_TIME placed in the *type* field of the *utmp* structure. The second record specifies the new date and is denoted by the string **new time** placed in the *line* field and the flag NEW_TIME placed in the *type* field. *wtmpfix* uses these records to synchronize all time stamps in the file.

In addition to modifying time and date stamps, *wtmpfix* will check the validity of the *name* field to ensure that it consists solely of alphanumeric characters or spaces. If it encounters an invalid name, it will change the login name to INVALID and write a diagnostic message to standard error. In this way, *wtmpfix* reduces the chance that *acctcon*(1M) will fail when processing connect-time accounting records.

FILES

/etc/wtmp login/logout history file

SEE ALSO

acct(1M), acctcon(1M), acctcms(1M), acctmerg(1M), acctprc(1M), acctsh(1M), runacct(1M).
acctcom(1) in the *CLIX Programmer's & User's Reference Manual*.
acct(2), acct(4), utmp(4) in the *UNIX System V Programmer's Reference Manual*.

NAME

gated - gateway routing daemon

SYNOPSIS

/etc/gated [-t[ierpuRH]] [*logfile*]

DESCRIPTION

gated is a routing daemon that handles multiple routing protocols and replaces *routed*(1M), *egpup*, and any routing daemon that speaks the HELLO routing protocol. *gated* currently handles the Routing Information Protocol (RIP), Exterior Gateway Protocol (EGP), and HELLO routing protocols. *gated* can be configured to perform all routing protocols or any combination of the three. The configuration for *gated* is by default stored in the **/etc/gated.conf** file.

Command-Line Tracing Options

gated can be invoked with a number of trace flags and an optional *logfile*. Tracing flags may also be specified in the configuration file with the **traceflags** clause. If tracing flags are specified without a log file, tracing output is sent to the controlling terminal. The valid trace flag prefix and possible modifiers associated with a trace flag are as follows:

-t If the **-t** prefix is used alone, log all error messages, route changes, and EGP packets sent and received. Using **-t** alone turns on the **i**, **e**, **r**, and **p** modifiers automatically. When **-t** is used with another modifier, the **-t** has no effect and only the accompanying modifiers are recognized. Note that **-t** must prefix other modifiers. The following modifiers are allowed.

- **i** Log all internal errors and interior routing errors.
- **e** Log all external errors due to EGP, exterior routing errors, and EGP state changes.
- **r** Log all routing changes.
- **p** Trace all EGP packets sent and received.
- **u** When used with **p**, **R**, **H**, or **N**, display the entire contents of routing packets sent and received.
- **R** Trace all RIP packets sent or received.
- **H** Trace all HELLO packets sent or received.

gated always logs fatal errors. If no *logfile* is specified and no tracing flags are set, all messages are sent to **/dev/null**.

Signal Processing

gated catches a number of signals and performs specific actions. Currently *gated* does special processing with the SIGHUP, SIGINT, and SIGUSR1 signals (see *signal*(2)).

When a SIGHUP is sent to *gated* and *gated* is invoked with trace flags and *logfile*, tracing is toggled off and the log file is closed. At this point the log file may be moved or removed. The next SIGHUP to *gated* will toggle the tracing on. *gated* reads the configuration file and sets the tracing flags to those specified with the **traceflags** clause. If no **traceflags** clause is specified, tracing is resumed using the trace flags specified on the command line. The *logfile* specified from the command line is created if necessary and the trace output is sent to that file. The trace output is appended to an already existing log file. This is useful for rotating log files.

Sending *gated* a SIGINT will cause a memory dump to be scheduled within the next 60 seconds. The memory dump will be written to the **/usr/tmp/gated_dump** file. *gated* will finish processing pending routing updates before performing the memory dump. The memory dump contains a snapshot of the current *gated* status, including the interface configurations, EGP neighbor status, and routing tables. If the **/usr/tmp/gated_dump** file already exists, the memory dump will be appended to the existing file.

When it receives a SIGUSR1, *gated* will reread selected information from the configuration file. This information currently includes the **announcetoAS**, **noannouncetoAS**, and **validAS** clauses. If no errors are detected, the new configuration information becomes effective. If errors are detected, the configuration information is not changed. *gated* will also check the interface status when it receives a SIGUSR1.

Configuration File Options Controlling Tracing Output

traceflags *traceflag* ...

The clause tells the *gated* process the level of tracing output desired. This option is read during *gated* initialization and when *gated* receives a SIGHUP. This option is overriden at initialization time if tracing flags are specified on the command line. The valid tracing flags are as follows:

internal	Log all internal errors and interior routing errors.
external	Log all external errors due to EGP, exterior routing errors, and EGP status changes.
route	Log all routing changes.
egp	Trace all EGP packets sent and received.
update	When used with **egp, rip, hello**, or **snmp**, display the contents of all routing packets sent and received.
rip	Trace all **rip** packets sent and received.
hello	Trace all **hello** packets sent and received.
icmp	Trace all **icmp** redirect packets received.
stamp	Print a timestamp to the log file every 10 minutes.
general	Use as a combination of **internal**, **external**, **route** and **egp**.
all	Enable all of the above tracing flags.

If more than one **traceflags** clause is used, the tracing flags accumulate since the trace flags are not mutually exclusive.

Default Configuration

gated normally reads configuration information from the **/etc/gated.conf** configuration file. If this file does not exist, *gated* assumes a default configuration file of the following:

RIP yes
HELLO no
EGP no

In addition, if the configuration file does not exist, there is only one network interface, and a default route is installed in the kernel, *gated* will exit, assuming that a simple default route is adequate.

Configuration File Options for Handling Routing

In this section, the numerous configuration options are explained. Each time the *gated* process is started, it reads the **/etc/gated.conf** file to obtain its instructions on how routing will be managed with respect to each protocol. The configuration options are as follows:

RIP {**yes**|**no**|**supplier**|**pointopoint**|**quiet**|**gateway #**}

This tells the *gated* process how to perform the RIP routing protocol. Only one of the above RIP arguments is allowed after the keyword **RIP**. If more than one is specified, only the first one is recognized. A list of the arguments to the **RIP** clause follows:

yes Perform the RIP protocol. Process all incoming RIP packets and supply RIP information every 30 seconds only if there are two or more network interfaces.

no Do not perform the RIP protocol.

supplier Perform the RIP protocol. Process all incoming RIP packets and force RIP information to be supplied every 30 seconds despite the number of network interfaces present.

pointopoint Perform the RIP protocol. Process all incoming RIP packets and force RIP information to be supplied every 30 seconds despite the number of network interfaces present. When this argument is specified, RIP information will not be sent out in a broadcast packet. The RIP information will be sent directly to the gateways listed in the **sourceripgateways** option described below.

quiet Process all incoming RIP packets, but do not supply any RIP information despite the number of network interfaces present.

gateway # Process all incoming RIP packets, supply RIP information every 30 seconds, and announce the default route

(0.0.0.0) with a metric of #. The metric should be specified in a value that represents a RIP hopcount. With this option set, all other default routes coming from other RIP gateways will be ignored. The default route is announced only when actively peering with at least one EGP neighbor and therefore should be used only when EGP is used.

If no RIP clause is specified, RIP will not be performed.

HELLO {**yes**|**no**|**supplier**|**pointopoint**|**quiet**|**gateway #**}
This tells *gated* how to perform the HELLO routing protocol. The arguments parallel the RIP arguments but do have some minor differences. Only one of the above HELLO arguments is allowed after the keyword **HELLO**. If more than one is specified, only the first one is recognized. A list of the arguments to the **HELLO** clause follows:

yes Perform the HELLO protocol. Process all incoming HELLO packets and supply HELLO information every 15 seconds only if there are two or more network interfaces.

no Do not perform the HELLO protocol.

supplier Perform the HELLO protocol. Process all incoming HELLO packets and force HELLO information to be supplied every 15 seconds despite the number of network interfaces present.

pointopoint Perform the HELLO protocol. Process all incoming HELLO packets and force HELLO information to be supplied every 15 seconds despite the number of network interfaces present. When this argument is specified, HELLO information will not be sent out in a broadcast packet. The HELLO information will be sent directly to the gateways listed in the **sourcehellogateways** option described below.

quiet Process all incoming HELLO packets, but do not supply any HELLO information despite the number of network interfaces present.

gateway # Process all incoming HELLO packets, supply HELLO information every 15 seconds, and announce the default route (0.0.0.0) with a time delay of #. The time delay should be specified in milliseconds. The default route is announced only when actively peering with at least one EGP neighbor. Therefore, it should be used only when EGP is running.

If no HELLO clause is specified, HELLO will not be performed.

EGP {**yes**|**no**}
This clause allows EGP processing by *gated* to be turned on or off.

no Do not perform EGP processing.

yes Perform all EGP operations.

By default, EGP processing will occur. Therefore, if no EGP clause is specified, all EGP operations will occur.

autonomoussystem #
If EGP protocol is being performed, this clause must be used to specify the autonomous system number (#). If not specified, *gated* will exit and give a fatal error message.

egpmaxacquire #
If EGP protocol is being performed, this clause specifies the number of EGP peers with which *gated* will be performing EGP. This number must be greater than 0 and less than or equal to the number of EGP neighbors specified or *gated* will exit. If this clause is omitted, all EGP neighbors will be acquired.

egpneighbor *gateway1* [**metricin** *metric*] [**egpmetricout** *egpmetric1*] [**ASin** *asin*] [**ASout** *asout*] [**AS** *as*] [**nogendefault**] [**acceptdefault**] [**defaultout** *egpmetric2*] [**validate**] [**intf** *interface*] [**sourcenet** *net*] [**gateway** *gateway2*]
If EGP protocol is being performed, this clause specifies the peers with which *gated* will be performing EGP. The neighbor, *gateway1*, can be either a symbolic name in **/etc/hosts** or an Internet Protocol (IP) hostname in Internet dot (n.n.n.n) notation. Dot notation is recommended to avoid confusion. Each EGP neighbor will be acquired in the order listed in the configuration file.

The **metricin** option specifies the internal time delay to be used as a metric for all of the routes learned from *gateway1*. *metric* should be specified as a time delay from 0-30000. If this option and the **validate** option are not used, the internal *metric* used is the EGP distance multiplied by 100.

The **egpmetricout** option specifies the EGP distance used for all networks advertised to *gateway1*. *egpmetric1* should be specified as an EGP distance in the range of 0-255. If this option is not specified, the internal time delay for each route will be converted to an EGP distance by division by 100, with distances greater than 255 being set to 255.

The **ASin** option verifies the autonomous system number *asin* of this neighbor. If the autonomous system number specified in neighbor acquisition packets does not verify, an error message is generated refusing the connection. If this option is not specified, autonomous system numbers are not verified.

The **ASout** option specifies the autonomous system number *asout* in EGP packets sent to *gateway1*. If not specified, the autonomous system specified in the **autonomoussystem** clause is used. This clause should not normally be used. It is reserved for a special situation interfacing between the Advanced Research Project Agency Network (ARPANET) and National Science Foundation Network (NSFNET).

The **AS** option specifies the autonomous system number that will be assigned to routes learned from *gateway1*. If *as* is not specified, the autonomous system used in the EGP packets received from this neighbor will be used. This clause should not normally be used. It is reserved for a special situation interfacing between the ARPANET and NSFNET.

The **nogendefault** option specifies that this neighbor should not be considered for the internal generation of a default when RIP gateway or HELLO gateway is used. If not specified, the internal default will be generated when actively peering with this neighbor.

The **acceptdefault** option is used to specify that the default route (network 0.0.0.0) should be valid when received from *gateway1*. If this option is not specified, the reception of the default route will cause a warning message to be printed and the route to be ignored.

The **defaultout** option specifies that the internally generated default may be passed to *gateway1* at the specified distance. The distance should be specified as an EGP distance from 0-255. A default route learned from another gateway will not be propagated to an EGP neighbor. Normally, EGP will not pass a default route. The **acceptdefault** option should not be specified when the **defaultout** option is used. *egpmetric1* specified in the **egpmetricout** option does not apply. The default route will always use *egpmetric2* specified by the **defaultout** option.

The **validate** option specifies that all networks received from *gateway1* must be specified in the **validAS** clause that also specifies this neighbor's autonomous system. Networks without a **validAS** clause will be ignored after a warning message is printed.

The **intf** option specifies *interface* used to send EGP packets to *gateway1*. This option is required only when no common net/subnet is with this EGP neighbor. This option currently is present only for testing purposes and does not imply correct operation when peering with an EGP neighbor that does not share a common net/subnet.

The **sourcenet** option specifies the source *net* to be specified in EGP poll packets sent to *gateway1*. If this option is not specified, the network (not subnet) of the interface used to communicate with *gateway1* is used. This option is currently present only for testing purposes and does not imply correct operation when used.

The **gateway** option specifies *gateway2* to be used when installing routes learned from an EGP neighbor on a different network. Normally these routes would be ignored. This option is currently present only for testing purposes and correct operation cannot be assured when it is used.

Configuration File Options For Handling Routing

The following configuration file options tell *gated* how to process both incoming and outgoing routing information:

trustedripgateways *gateway* ...
trustedhellogateways *gateway* ...

When these clauses are specified, *gated* will listen only to RIP or HELLO information, respectively, from these RIP or HELLO gateways. *gateway* can be either a symbolic name from **/etc/hosts** or an IP host address in dot notation (n.n.n.n). Again, dot notation is recommended to eliminate confusion. This clause does not restrict the propagation of routing information.

sourceripgateways *gateway* ...
sourcehellogateways *gateway* ...

gated will send RIP or HELLO information directly to *gateways* specified. If **pointopoint** is specified in the **RIP** or **HELLO** clauses, *gated* will send only RIP or HELLO information to specified *gateways*. *gated* will not send any information using the broadcast address. If **pointopoint** is not specified in those clauses and *gated* is supplying RIP or HELLO information, *gated* will send information to specified *gateways* and broadcast it using a broadcast address.

noripoutinterface intf *addr* [*intfaddr*] ...
nohellooutinterface intf *addr* [*intfaddr*] ...
noripfrominterface intf *addr* [*intfaddr*] ...
nohellofrominterface intf *addr* [*intfaddr*] ...

The above clauses turn protocols on and off for each interface *addr*. **no{rip|hello}frominterface** means that no RIP or HELLO information will be accepted into the listed interfaces from another gateway. **no{rip|hello}outinterface** means that no RIP or HELLO knowledge will be sent from the listed interfaces. The *intfaddr* should be in dot notation (n.n.n.n.).

passiveinterfaces intf *addr* [*intfaddr*] ...

In order to dynamically determine if an interface is properly functioning, *gated* will time out an interface when no RIP, HELLO, or EGP packets are being received on that particular interface. Packet Switch Network (PSN) interfaces send a RIP or HELLO packet to themselves to determine if the interface is properly functioning as the delay between EGP packets may be longer than the interface timeout. Routes for interfaces that have timed out automatically are reinstalled when routing information is again received over the interface. The above clause stops *gated* from timing out the listed interfaces.

The interfaces listed will always be up and working. If *gated* is not a RIP or HELLO supplier, all interfaces will not be aged and the **passiveinterfaces** automatically applies to all interfaces.

interfacemetric *intfaddr metric#*

This feature allows an interface metric to be specified for the listed interface. On systems that support interface metrics, this clause will override the kernel's metric. On systems that do not support an interface metric, this feature allows one to be specified. The interface metric is added to the true metric of each route that comes in through routing information from the listed interface. The interface metric is also added to the true metric of any information sent out through the listed interface. The metric of directly attached interfaces is also set to the interface metric. Routing information broadcast about directly attached networks will be based on the interface metric specified. This clause is required for each interface on which an interface metric is desired.

reconstmetric *intfaddr metric#*

This is a first attempt to support fallback routing in *gated*. If the above clause is used, the metrics of the routes contained in any RIP information coming into the listed interface will be set to the specified *metric#*. Metric reconstitution should not be used lightly, since it could be a major contributor in forming routing loops. Use this with extreme caution. Any route that has a metric of infinity will not be reconstituted and will remain infinity.

fixedmetric intfaddr proto {**rip**|**hello**} *metric#*

This is another attempt to support fallback routing in *gated*. If the above clause is used, all routing information sent out the specified interface will have a metric of *metric#*. For RIP, specify the metric as a RIP hopcount from 0 to infinity. For HELLO, specify the metric as a HELLO delay in milliseconds from 0 to infinity. Any route that has a metric of infinity will remain infinity. Fixed metrics should also be **used with extreme caution.**

donotlisten *net* **intf** *addr* ... **proto** {**rip**|**hello**}
donotlistenhost *host* **intf** *addr* ... **proto** {**rip**|**hello**}

This clause reads as follows: keyword **donotlisten** followed by a network number, which should be in dot notation followed by keyword **intf**. Then a list of interfaces in dot notation precede the keyword **proto**, followed by **rip** or **hello**.

This means that any information regarding *net* coming in through the specified protocols and from the specified interfaces will be ignored. The keyword **all** may be used after the keyword **intf** to specify all interfaces on the machine. Consider the following example:

donotlisten 10.0.0.0 intf 128.84.253.200 proto rip

This means that any RIP information about network 10.0.0.0 coming in through interface 128.84.253.200 will be ignored. One clause is required for each network on which this restriction is desired.

donotlisten 26.0.0.0 intf all proto rip hello

This means that any RIP and HELLO information about network 26.0.0.0 coming in through any interface will be ignored.

donotlistenhost can be described the same way as above except that a host address is provided instead of a network address. Restrictions of the nature described above are applied to the specified host route the specified routing protocol learns about.

listen *net* **gateway** *addr* ... **proto** {**rip|hello**}
listenhost *host* **gateway** *addr* ... **proto** {**rip|hello**}

This clause reads as follows: keyword **listen** followed by a network number that should be in dot notation followed by keyword **gateway**. Then a list of gateways in dot notation should precede the keyword **proto**, followed by **rip** or **hello**.

This means to listen only to information about network **net** by the specified protocol(s) only from the listed **gateways**. Consider the following example:

listen 128.84.0.0 gateway 128.84.253.3 proto hello

This means that any HELLO information about network 128.84 coming in through gateway 128.84.253.3 will be accepted. Any other information about 128.84 from any other gateway will be rejected. One clause is necessary for each network to be restricted.

listenhost 26.0.0.15 gateway 128.84.253.3 proto rip

This means that any information about host 26.0.0.15 must come by RIP and from gateway 128.84.253.3. All other information regarding this host will be ignored.

announce *net* **intf** *addr* ... **proto** *type* [**egpmetric #**]
announcehost *host* **intf** *addr* ... **proto** *type* [**egpmetric #**]
noannounce *net* **intf** *addr* ... **proto** *type* [**egpmetric #**]
noannouncehost *host* **intf** ... **proto** *type* [**egpmetric #**]

These clauses restrict networks and identify the protocols by which they are restricted. The **announce[host]** and **noannounce[host]** clauses may not be used together on the same interface. With the **announce[host]** clause, *gated* will announce only the networks or hosts that have an associated **announce[host]** clause with the appropriate protocol. With the **noannounce[host]** clause, *gated* will announce everything except networks or hosts that have an associated **noannounce[host]** clause. These clauses allow a choice of announcing only what is on the announce list or everything except networks on the **noannounce** list on a per-interface basis.

The arguments are the same as the arguments in the **donotlisten** clause except **egp** may be specified in the *proto* field. *type* can either be **rip, hello, egp,** or any combination of the three. When **egp** is specified in the *proto* field, an egp metric must be specified. This is the metric at which *gated* will announce the listed network through EGP.

These are not static route entries. These restrictions will apply only if the network or host is learned by one of the routing protocols. If a restricted network suddenly becomes unreachable and goes away, announcement of this network will stop until it is learned again.

Currently, only one **announce[host]** or **noannounce[host]** may be specified per network or host. It is not possible to announce a network or host through HELLO out one interface and through RIP out another.

Consider the following examples:

```
announce 128.84 intf all proto rip hello egp egpmetric 0
announce 10.0.0.0 intf all proto rip
announce 0.0.0.0 intf 128.84.253.200 proto rip
announce 35.0.0.0 intf all proto rip egp egpmetric 3
```

With only these four **announce** clauses in the configuration file, *gated* will announce only these four networks. It will announce 128.84.0.0 by RIP and HELLO to all interfaces and announce it by EGP with a metric of 0. RIP will announce network 10.0.0.0 to all interfaces. RIP will announce network 0.0.0.0 (default) out interface 128.84.253.200 only. RIP will announce network 35.0.0.0 to all interfaces and EGP will announce it with a metric of 3. These are the only networks that will be broadcast by this gateway. Once the first **announce** clause is specified, only the networks with **announce** clauses will be broadcast; this includes local subnetworks.

Once an **announce[host]** or **noannounce[host]** has an **all** specified after an **intf**, that clause is applied globally and the option of having per-interface restrictions is lost. If no routing announcement restrictions are desired, **announce** clauses should not be used. All information learned will then be propagated out. These clauses do not affect the information to which *gated* listens. Any network that does not have an **announce** clause is still added to the kernel routing tables, but none of the routing protocols announce the network. To stop networks from being added to the kernel, the **donotlisten** clause may be used.

```
announce 128.84 intf 128.59.2.1 proto rip
noannounce 128.84 intf 128.59.1.1 proto rip
```

The above clauses mean that on interface 128.59.2.1, RIP will announce only information about 128.84.0.0, but on interface 128.59.1.1, RIP will announce all information except 128.84.0.0.

noannounce 128.84 intf all proto rip hello egp egpmetric 0
noannounce 10.0.0.0 intf all proto hello

These clauses mean that except for the two specified networks, all networks will be propagated. Specifically, no protocol will announce network 128.84.0.0 on any interface. Knowledge of 128.84.0.0 is not sent anywhere. HELLO will not announce network 10.0.0.0 to any interface. This also implies that RIP will announce network 10.0.0.0 to every interface. EGP will also broadcase this network with a metric specified in the **defaultegpmetric** clause.

defaultegpmetric #

This is a default EGP metric to use when there are no routing restrictions. Normally, with no routing restrictions, *gated* announces all networks learned by HELLO or RIP by EGP with this specified default EGP metric. If this clause is not used, the default EGP metric is set to 255, which would ignore any EGP advertised route of this nature. When there are no routing restrictions, any network with a direct interface is announced by EGP with a metric of 0. The announcements do not include subnets. It includes only the nonsubnetted network.

defaultgateway *gateway proto* [*metric*] {**active|passive**}

This default gateway is installed in the kernel routing tables during initialization and is reinstalled when information about the default route is lost. This route is installed with the time delay equivalent of a RIP metric of 15 unless another metric is specified with the **metric** option.

If **RIP** gateway or **HELLO** gateway are in use, this default route is deleted when successfully peering with an EGP neighbor not specified for **nogendefault**.

Any other default route learned by another routing protocol will override an **active** default route. Only a default route with a lower metric will override a **passive** default route.

An **active** default route will not be propagated in routing updates; a **passive** default route will be propagated.

gateway should be an address in dot notation. *metric* is optional and should be a metric in the specified protocol between zero and infinity. If not specified, a RIP metric of 15 is used. The *proto* field should be either **rip**, **egp**, or **hello**. The *proto* field initializes the protocol by which the route was learned.

net *netaddr* **gateway** *addr* **metric** *hopcnt* {**rip|egp|hello**}
host *hostaddr* **gateway** *addr* **metric** *hopcnt* {**rip|egp|hello**}

The following clauses install a static route to **net** *netaddr* or **host** *hostaddr* through gateway *addr* at a metric of *hopcnt* learned by either RIP, HELLO, or EGP. If *hopcnt* is 0, *netaddr* is logically equivalent to the network on the interface with address *addr*. As usual, dot notation is recommended for the addresses. This route will be installed in the kernel's routing table and will never be affected by any other gateway's RIP or HELLO announcements. The protocol by which it was learned is important if EGP will announce the route. If the protocol is **rip** or **hello** and there are no routing restrictions, EGP will announce the route with a metric specified in the **defaultegpmetric** clause. If the protocol is **egp** and there are no routing restrictions, EGP will announce route with a metric of *hopcnt*.

egpnetsreachable *net* ...

This option remained as a soft restriction. It cannot be used when the **announce** or **noannounce** clause is used. Normally, with no restrictions, *gated* announces all routes learned from RIP and HELLO by EGP. The **egpnetsreachable** clause restricts EGP announcement to the networks listed in the clause. The metric used for the HELLO and RIP learned routes is the value given in the **defaultegpmetric** clause. If this clause does not specify a value, the value is set to 255. With the **egpnetsreachable** clause, individual unique EGP metrics may not be set for each network. The **defaultegpmetric** is used for all networks except those that are directly connected, which use a metric of 0.

martiannets *net* ...

This clause appends to *gated*'s list of martian networks. Martian networks are those known to be invalid and should be ignored. When *gated* learns of one of these networks through any means, it will immediately ignore the network. If **external** tracing is enabled, a message will be printed to the trace log. Multiple occurrences of the **martiannets** clause accumulate.

An initial list of martian networks is coded into *gated* in the include file **rt_control.h**. This list contains 127.0.0.0, 128.0.0.0, 191.253.0.0, 192.0.0.0, 223.255.255.0, and 224.0.0.0.

Configuration File Options for Autonomous System (AS) Routing

In the internal routing tables, *gated* maintains the autonomous system number from which each route was learned. Autonomous systems are used only when an exterior routing protocol is in use (in this case EGP). Routes are tagged with the autonomous system number of the EGP peer from which they were learned. Routes learned by the interior routing protocols, RIP and HELLO, are tagged with the autonomous system number specified in the **autonomoussystem** clause.

gated normally does not propagate routes learned from exterior routing protocols to interior routing protocols. Historically this is because of the ARPANET core EGP speakers that do not have adequate validation of routing information they receive. Some of the following clauses allow exterior routes to be propagated by interior protocols. Therefore, it is crucial for the user to be extremely cautious when allowing exterior routes to be propagated. They should not be used unless their authors are consulted if the user is in doubt about their use.

The following clauses provide limited control over routing based on autonomous system number.

validAS *net* **AS** *as* **metric** *metric*

The **validAS** clause validates networks from certain autonomous systems. When an EGP update is received from a neighbor that has the **validate** option specified on the associated **egpneighbor** clause, a **validAS** clause is searched for, specifying the newly received network and the autonomous system number of the EGP neighbor. If the appropriate **validAS** clause is located, the network is considered for addition to the routing table with the specified metric. If a **validAS** clause is not located, a warning message is printed and the network is ignored.

A network may be specified in several **validAS** clauses as being associated with several different autonomous systems.

announcetoAS *as0* {**restrict**|**norestrict**} **ASlist** *as1* ...
noannouncetoAS *as0* {**restrict**|**norestrict**} **ASlist** *as1*...

The **announcetoAS** and **noannouncetoAS** control the exchanging of routing information between different autonomous systems. Normally *gated* will not propagate routing information between autonomous systems. The exception to this is that routes learned from *gated*'s own autonomous system by RIP and HELLO will be propagated by EGP. These clauses allow information learned by EGP from one autonomous system to be propagated by EGP to another autonomous system or by RIP and HELLO to *gated*'s own autonomous system.

If the **announcetoAS** clause is specified, information learned by EGP from autonomous systems *as1* ... will be propagated to autonomous system *as0*. If *gated*'s own autonomous system, as specified in the **autonomoussystem** clause, is specified as *as0*, RIP and HELLO will propagate this information. Routing information from autonomous systems not specified in the AS list will not be propagated to autonomous system *as0*.

If the **noannouncetoAS** clause is specified, information learned by EGP from all autonomous systems except *as1* ... will be propagated to autonomous systems *as0*. If *gated*'s own autonomous system is specified as *as0*, this information will not be propagated by RIP and HELLO.

The **[no]restrict** option controls the application of **announce** and **noannounce** clauses to the propagation of routes to different autonomous systems. If **restrict** is specified, normal announcement restrictions apply. If **norestrict** is specified, announcement restrictions are not considered. All routes from the source autonomous systems are propagated to the destination autonomous system.

Only one **announcetoAS** or **noannounceAS** clause may be specified per target autonomous system.

Notes on Configuration Options

gated stores its process ID in the **/etc/gated.pid** file.

If EGP is being used when supplying the default route (through RIP gateway or HELLO gateway) and all EGP neighbors are lost, the default route will not be advertised until at least one EGP neighbor is regained.

With the complexity of the current network topology and with many back-door paths to networks, the use of routing restrictions is recommended. With the current routing strategies, it is easy for illegal or invalid networks to penetrate into the ARPANET core or the NSFNET backbone. Using routing restrictions takes a little more maintenance time and routing restrictions are not the long-term answer, but for now they must be used.

Gated Internal Metrics

gated stores all metrics internally as a time delay in milliseconds to preserve the granularity of HELLO time delays. The internal delay ranges from 0 to 30000 milliseconds, with 30000 representing infinity. Metrics from other protocols are translated to and from a time delay as they are received and transmitted. EGP distances are not comparable to HELLO and RIP metrics but are stored as a time delay internally to compare with other EGP metrics. The conversion factor between EGP distances and time delays is 100. RIP and interface metrics are translated to and from the internal time delays with the following translation tables:

Time Delay	RIP Metric	RIP Metric	Time Delay
0 - 0	0	0	0
1 - 100	1	1	100
101 - 148	2	2	148
149 - 219	3	3	219
220 - 325	4	4	325
326 - 481	5	5	481
482 - 713	6	6	713
714 - 1057	7	7	1057
1058 - 1567	8	8	1567
1568 - 2322	9	9	2322
2323 - 3440	10	10	3440
3441 - 5097	11	11	5097
5098 - 7552	12	12	7552
7553 - 11190	13	13	11190
11191 - 16579	14	14	16579
16580 - 24564	15	15	24564
24565 - 30000	16	16	30000

Notes on Implementation Specifics

In the *gated* configuration file, all references to Point-to-Point (PTP) interfaces must use the destination address. This is the only change made to the configuration file syntax from earlier versions, which used the source address of the PTP link. Otherwise, old configuration files should be compatible.

All protocols have a two-minute hold down. When a routing update indicates that the route in use is being deleted, *gated* will not delete the route for two minutes.

Changes can be made to the interfaces and *gated* will notice them. The *gated* process does not need to be restarted. If the netmask, subnetmask, broadcast address, or interface metric is changed, the interface should be marked down with *ifconfig*(1M) and then marked up at least 30 seconds later. Flag changes do not require the interface to be brought down and back up.

To handle PTP links more consistently, RIP propagates and listens to host routes. This version also supports the RIP_TRACE commands.

Subnet interfaces are supported. Subnet information will be propagated only on interfaces to other subnets of the same network. For example, if there is a gateway between two class B networks, the subnet routes for each respective class B network are not propagated into the other class B network. Only the class B network number is propagated.

gated listens to host and network REDIRECTs and tries to take an action on the REDIRECT for its own internal tables that parallels the kernel's action. In this way, the redirect routine in *gated* parallels the Berkeley kernel redirect routine as closely as possible. Unlike the Berkeley kernel, *gated* deletes routes learned by a REDIRECT after six minutes. The route is then deleted from the kernel routing tables. This helps keep the routing tables more consistent. Any route that was learned by a REDIRECT is **not** announced by any routing protocol.

The *gated* EGP code verifies that all networks sent and received are valid class A, B, or C networks according to the EGP specification. Information about networks that do not meet these criteria is not propagated. If an EGP update packet contains information about a network that is not either class A, B, or C, the update is in error and is ignored. Only the information about the specific network will be ignored if *gated* is compiled with the EGP_IGNORE_BAD define specified. This option should be used with caution.

FILES

/etc/gated.conf	configuration file
/etc/gated.pid	process-id of the running *gated*
/usr/tmp/gated_dump	memory dump file
/etc/gated.version	*gated* version information

SEE ALSO

routed(1M).

NAME

getinet - Internet address generation utility

SYNOPSIS

/usr/ip32/inc/getinet

DESCRIPTION

getinet is an interactive utility that assists in the assignment of an Internet address for the CLIX node on which it is executed. Super-user privilege is required.

If an Internet address has already been assigned to the node, *getinet* exits with no action. If no address has been assigned, *getinet* displays a menu that presents the user with the following choices:

e Prompt the user for the Internet address to be used for this node.

g Generate a class A Internet address based on the node's Ethernet hardware address.

q Quit *getinet* without assigning an Internet address to the node.

getinet must be used with caution. Incorrect assignment of Internet addresses can cause serious problems on a network. If *getinet* is allowed to generate the address (**g** option) of any node on a network, it must be allowed to do so for all nodes on that network. It should not be used to generate addresses on any network, which includes non-CLIX nodes.

The format of the address entered in response to the prompt of the **e** option is checked for adherence to the Internet format. A correct format does not necessarily mean that the address semantics are correct.

SEE ALSO

"BSD Network Configuration Tutorial" in the *CLIX System Guide*.

NAME

ifconfig - configure network interface parameters

SYNOPSIS

/etc/ifconfig *interface-name* [*address-family* [*address*]] [*parameters*]

DESCRIPTION

ifconfig is the network interface configuration utility. *ifconfig* is run at boot time to define the Department of Defense (DoD) Internet address as well as other attributes for each physical network interface configured to use the DoD Internet Protocol (IP) suite.

interface-name is a string consisting of the device name of an interface device, such as **/dev/et0**.

The default value of *address-family* is **inet**, the Internet Protocol family, which is the only address family currently supported. *Address* may be a host name present in **/etc/hosts** or a DoD Internet address in standard dot notation, such as 129.135.200.7.

The following are valid *parameters*:

up Mark an interface as "up" (enable transmission and reception of Internet datagrams).

down Mark an interface as "down" (disable transmission and reception of Internet datagrams).

metric *n*
Set the routing metric of the interface to *n*. The default is one. The routing metric is used in determining the number of hops a datagram may take before reaching its final destination.

netmask *mask*
Specify the portion of the Internet address to reserve for dividing networks into subnetworks. *Mask* is a 32-bit mask in which the set bits specify the bits of the Internet address to be treated as the network and subnetwork fields. The subnetwork field consists of bits that would ordinarily, according to the address class definition, belong to the host field. Clear bits in *mask* represent the host field. *Mask* can be specified as a single hexadecimal number with a leading 0x, as an Internet address in standard dot notation, or with a network name listed in the network database, **/etc/networks**. Ordinarily the subnetwork mask is set automatically to the value returned in response to an ICMP Address Mask Request issued on the physical network. If no response is received, manual configuration using *ifconfig* is necessary.

broadcast *bcast_addr*
Specify the address to be used as the broadcast address on the network. The default value of *bcast_addr* is the network portion of the local address with all ones for the host portion.

maskrep
Authorize this host to reply to ICMP address mask requests received on this interface.

-maskrep
Disallow this host from replying to ICMP address mask requests.

If *parameters* are not supplied, *ifconfig* displays the current configuration of *interface-name.*

Messages indicating that the specified interface does not exist, the requested address is unknown, the given parameter is not supported, or the user is not privileged to perform the specified operation are displayed if such conditions are encountered.

SEE ALSO

inet(7B), ip(7S).
inet(3B), hosts(4), networks(4) in the *CLIX Programmer's & User's Reference Manual.*

NOTES

Only the super-user may modify the configuration of a network interface.

NAME

incd - Intergraph Network Configuration daemon

SYNOPSIS

/etc/incd

DESCRIPTION

incd configures the desired network protocols on physical network interfaces. Desired network protocols to be configured for an interface are specified in the **/etc/incd.conf** file. The format for each line in this file is as follows:

> *interface_name* [**xns**] [**dod** [**arp**] [**trlr**] [**udp**] [**tcp**]]

The *interface_name* parameter is the name of the interface device (for example, **et0**).

The valid protocol configuration options are as follows:

xns Indicates that the Xerox Network System (XNS) protocols are desired on this interface.

dod Indicates that the specified Department of Defense (DoD) Internet protocols are desired on this interface. The DoD protocols will not be configured unless an entry exists in the **/etc/hosts** file for the node name of the host. Specific portions of the DoD Internet protocol suite which are desired must be specified.

Internet protocol options are as follows:

arp Address Resolution Protocol

trlr Reception of trailer encapsulated packets

udp User Datagram Protocol

tcp Transmission Control Protocol

There should be a single configuration line for each network interface with the fields separated by blanks and/or tabs. The *interface_name* parameter for each interface should begin in column one. Comment lines in the configuration file are denoted by a **#** in column one.

SEE ALSO

arp(7B), ifconfig(1M), ip(7S), tcp(7B), tcp(7S), et(7S), inet(7B), udp(7B), udp(7S).
hosts(4) in the *CLIX Programmer's & User's Reference Manual.*
"Network Programming Tutorial" in the *CLIX System Guide.*

DIAGNOSTICS

If an error occurs during configuration, a message indicating the error is printed to **stderr**. If the error is serious enough to prevent the configuration of all specified protocols, *incd* exits with a nonzero status. Otherwise, *incd* continues execution.

NAME

incmon - Intergraph Network Core XNS monitor

SYNOPSIS

incmon [*node* | [*lan.*]*addr*] [**/***option*] [**/ti**[**meout**]=*seconds*] [**/po**[**rt**]=*n*]

incmon [**/no**[**dename**]=*node*] [**/***option*] [**/ti**[**meout**]=*seconds*] [**/po**[**rt**]=*n*]

incmon [[**/ne**[**twork**]=*lan*] **/ad**[**dress**]=*addr*] [**/***option*] [**/ti**[**meout**]=*seconds*] [**/po**[**rt**]=*n*]

DESCRIPTION

incmon, the Intergraph Network Core monitor, is a network monitor utility that allows a machine's performance to be viewed on a Xerox Network Systems (XNS) network. By viewing several machines on the network, the user is able to diagnose problems occurring or access information relative to the current network loading and performance. *incmon* is a menu-driven utility that can be used only on VT52, VT100, VT200, VT220, or VT300 compatible terminals.

If a *node*, an Ethernet address (*addr*), or an Ethernet address and a Local Area Network (LAN) number (*lan*) is not specified, *incmon* will default to the local machine's network address. From *incmon*'s main menu, the network number, address, or node name can be changed.

The **/ti**[**meout**]=*seconds* option on the command line specifies the request timeout in seconds. The timeout value has a maximum limit of 1200 seconds (20 minutes). The minimum timeout is one second and the default timeout is 20 seconds.

The **/po**[**rt**]=*n* option on the command line specifies the remote router port to be used in general statistics. The *n* is **x** or **t** for the transceiver port and 0-7 for one of the other ports. By default, the general statistics for a remote router will be for the port on which the router received the network request for statistics.

When *incmon* is specified without an *option*, a main menu is displayed. From the main menu, one of several displays may be chosen. If an *option* is present on the command line, *incmon* will begin with a display corresponding to one of the following options:

/ge[**neral**]	General Statistics Display
/lo[**opback**]	Loopback Test Utility Display
/se[**arch**]	Network Search Report Display
/ro[**uting**]	Router Information Display
/to[**pology**]	XNS Network Topology Display

General Statistics Display

This option displays statistics that tell how the specified *node* is functioning on the network. Information displayed includes error and status counters, software image name and version, and the number of bytes and frames transmitted and received and their effective rates. The information on this display is updated continually.

Loopback Test Display
This option allows the user to transmit and receive frames through the XNS ECHO protocol. This display contains options that control the test frame, change the frame buffer size, alter the timeout and time delay between transmissions, or perform a software checksumming test on the frame.

Network Search Report Display
This option allows the user to view the contents of a database on the network that contains basic general information on all nodes on the network.

Router Information Display
This option shows information about the local network's XNS routing topology. It shows the remote routers that the specified *node* can access and the LANs the *node* can access through the remote Routers. The menu also indicates the relative distance between the specified node and a particular LAN and can tell exactly what is connected to a particular remote router's ports.

XNS Network Topology Display
This option shows network topology using a tree structure that starts from the specified node's local LAN. The display includes Remote Routers and LANs that can be reached from the specified node.

SEE ALSO

incmond(1M).

NOTES

An XNS socket is a physical Ethernet port and not a *socket*(2B).

NAME

incmond - Intergraph Network Core XNS monitor daemon

SYNOPSIS

/usr/ip32/inc/incmond [-l *logfile*] [-v]

DESCRIPTION

incmond is the Xerox Network Services (XNS) monitor daemon that executes at system startup on Intergraph workstations and servers to service requests from the Intergraph Network Core monitor, *incmon*(1M). *incmond* manages a periodic multicast of an informational frame about the node's identity. *incmond* running on 300-series, 400-series, 3000-series, 4000-series, and 6000-series workstations/servers will collect and maintain this information for all Intergraph nodes on the local area network. *incmond* also retrieves the node's XNS network routing database and networking statistics that are maintained by the node's kernel.

The following options are available:

-l *logfile* Log errors to *logfile*.

-v Display the version number of *incmond* without invoking the daemon process.

SEE ALSO

incmon(1M).

NAME

inetd - Internet "super server"

SYNOPSIS

/usr/ip32/tcpip/inetd [**-d**] [*configuration-file*]

DESCRIPTION

inetd is run at boot time to listen for connections on certain Internet sockets. When a connection is found on one of its sockets, *inetd* invokes a program to service the request and continues to listen on the socket (except in the cases described below). Essentially, *inetd* allows running one daemon to invoke several others, reducing system load.

On execution, *inetd* reads its configuration information from a configuration file which, by default, is **/etc/inetd.conf**. The fields of the configuration file are as follows:

service name
socket type
protocol
[**wait** | **nowait**]
user
server program
server program arguments

Each field must have an entry. The *service name* entry is the name of a valid service in the file **/etc/services**. For "internal" services (discussed below), the service name must be the official name of the service (that is, the first entry in **/etc/services**).

The *socket type* may be **stream** or **dgram**, depending on whether the socket is a stream or datagram socket.

The *protocol* must be a valid protocol as given in **/etc/protocols** ("tcp" and "udp" are examples).

The **wait/nowait** entry is applicable to datagram sockets only. (Other sockets should have a **nowait** entry in this space.) If a datagram server connects to its peer, freeing the socket so *inetd* can receive further messages on the socket, it is a "multithreaded" server, and contains the **nowait** entry. For datagram servers that process all incoming datagrams on a socket and eventually time out, the server is "single-threaded" and should use a **wait** entry. If a datagram establishes pseudo connections, it must be listed as **wait** to avoid a race. In this case, the server reads the first packet, creates a new socket, and then forks and exits to allow *inetd* to check for new service requests to spawn new servers.

User contains the name of the user the server should run as. This allows servers to have less permission than root. The *server program* entry contains the path name of the program to be executed by *inetd* when a request is found on its socket. If *inetd* provides this service internally, this entry should be **internal**, and no server program arguments should be given.

The arguments to the server program begin with argv[0], which is the name of the program.

inetd provides several trivial services internally by routines within itself. These services are echo, discard, chargen (character generator), daytime (human readable time), and time (machine-readable time, in the form of the number of seconds since midnight, January 1, 1970). For details of these services, consult the appropriate RFC from the Network Information Center.

inetd rereads its configuration file when it receives the hangup signal, SIGHUP. Services may be added, deleted, or modified when the configuration file is reread.

SEE ALSO

ftpd(1M), rexecd(1M), rlogind(1M), rshd(1M), telnetd(1M), tftpd(1M).

NOTES

For security reasons, *tftpd*(1M) is not serviced by *inetd*.

NAME

jbexport - remove an optical disk platter from a jukebox

SYNOPSIS

jbexport *volume1 volume2*

DESCRIPTION

If a platter with volume names *volume1* and *volume2* is in any jukebox, that platter is ejected through the mail slot and all information about it is removed from the Jukebox Interface Management System (JIMS) database.

SEE ALSO

jbimport(1M).
jbconfig(1), odintro(1) in the *CLIX Programmer's & User's Reference Manual.*

NAME

jbimport - introduce a labeled optical disk platter to a jukebox

SYNOPSIS

jbimport *volume1 volume2 jukebox_id*

DESCRIPTION

jbimport causes the mail slot of the specified jukebox to open. When the operator inserts a platter in the slot, the Jukebox Interface Management System (JIMS) mounts it in an optical disk drive and verifies that the two sides are labeled as specified by the *volume1* and *volume2* arguments. If the volume names match, they are placed in the JIMS database and the platter is stored in the jukebox. If the names do not match, the volume will be ejected from the mail slot and the operator may try a different platter. Three attempts to obtain the correct platter are allowed. After three unsuccessful attempts, the command returns an error message. *Jukebox_id* is the logical jukebox name specified in the JIMS configuration file. This name is reported by *jbconfig*(1).

SEE ALSO

jbexport(1M).
jbconfig(1), JBCFG(4), odintro(1) in the *CLIX Programmer's & User's Reference Manual.*

NAME

jbinventory - take inventory for all optical disk platters in a jukebox

SYNOPSIS

jbinventory *jukebox_id*

DESCRIPTION

jbinventory constructs a Jukebox Interface Management System (JIMS) database that associates volume names with storage slot numbers in an optical disk jukebox so that volumes may be mounted by name. The labels of all platters in the jukebox are read. *Jukebox_id* is the logical jukebox name specified in the JIMS configuration file. This name is reported by *jbconfig*(1).

SEE ALSO

jbstart(1M).
jbconfig(1), JBCFG(4) in the *CLIX Programmer's & User's Reference Manual.*

WARNINGS

This process can require over an hour for a full jukebox.

NAME

jblabel - introduce an unlabeled optical disk platter to a jukebox and label it

SYNOPSIS

jblabel *volume1 volume2 jukebox_id*

DESCRIPTION

jblabel functions like *jbimport*(1M) except that *jblabel* assumes that the platter is unlabeled. *jblabel* initializes each volume in the same manner as *odlabel*(1M) using *volume1* and *volume2* as volume names. The volumes are entered in the Jukebox Interface Management System (JIMS) database and are ready for use.

Jukebox_id is the logical jukebox name specified in the JIMS configuration file. This name is reported by *jbconfig*(1).

SEE ALSO

odlabel(1M), jbimport(1M).
jbconfig(1), JBCFG(4) in the *CLIX Programmer's & User's Reference Manual.*

NAME

jbstart - initializes JIMS

SYNOPSIS

jbstart

DESCRIPTION

jbstart initializes the Jukebox Interface Management System (JIMS). Normally, *jbstart* does not need to be used. A shell script starts JIMS automatically at system startup. This script is installed when the product is delivered.

JIMS can perform either a cold or a warm start. *jbstart* prompts the user for the type of start to be performed (cold or warm). For a cold start, JIMS has no knowledge of the jukebox contents and performs a full inventory, reading the labels of all platters in the jukebox. This process can require over an hour for a full jukebox.

A warm start requires less time than a cold start because it does not perform a full inventory on the jukebox. JIMS records the jukebox's state when it was shut down on magnetic disk. When JIMS is started, it knows whether it was shut down properly. If it was shut down correctly, a warm start, consisting only of reading the magnetic disk record of the inventory, is sufficient. If JIMS was not shutdown correctly, a cold start is needed.

FILES

/usr/ip32/od/JBCFG

SEE ALSO

jbterminate(1), JBCFG(4).

NOTES

jbstart prompts for the JIMS configuration file. If a file is not supplied, **/usr/ip32/od/JBCFG** is used. This file contains logical names for the jukeboxes and its optical drives, terminal devices to which the jukebox robotics are connected, generic Small Computer System Interface (SCSI) character devices to which optical drives are connected, and optional modes within which JIMS can operate.

BUGS

JIMS releases encounter problems with warm starts. If platters cannot be mounted after a warm start, JIMS should be terminated first and then *jbstart* used to initiate a cold start.

NAME

jbterminate - perform an orderly shutdown of JIMS

SYNOPSIS

jbterminate

DESCRIPTION

jbterminate stops the Jukebox Interface Management System (JIMS). Normally, *jbterminate* does not need to be used. A shell script run at system shut down usually stops JIMS automatically. This script is installed when the product is delivered.

SEE ALSO

jbstart(1M).

NAME

jbvaryoff - remove a jukebox or drive from the current active configuration

SYNOPSIS

jbvaryoff *jukebox_id* [:*drive_id*]

DESCRIPTION

jbvaryoff removes a jukebox or drive from the current active configuration. Its parameters are the jukebox and/or drive name as reported by *jbconfig*(1).

Jukebox_id is the logical jukebox name specified in the Jukebox Interface Management System (JIMS) configuration file. This name is reported by *jbconfig*(1). *Drive_id* is the logical optical disk drive name as entered in the configuration file, **/usr/ip32/od/JBCFG**.

EXAMPLES

jbvaryoff JB01
jbvaryoff JB01:OD001

FILES

/usr/ip32/od/JBCFG

SEE ALSO

jbconfig(1), JBCFG(4) in the *CLIX Programmer's & User's Reference Manual.*

NAME

jbvaryon - add or return jukebox or drive to the current active configuration

SYNOPSIS

jbvaryon *jukebox_id* [:*drive_id*]

DESCRIPTION

jbvaryon adds or returns a jukebox or drive to the active configuration. Its parameters are the jukebox and/or drive names as reported by *jbconfig*(1).

Jukebox_id is the logical jukebox name specified in the Jukebox Interface Management System (JIMS) configuration file. This name is reported by *jbconfig*(1). *Drive_id* is the logical optical disk drive name as entered in the configuration file, **/usr/ip32/od/JBCFG**.

EXAMPLES

jbvaryon JB01
jbvaryon JB01:OD001

FILES

/usr/ip32/od/JBCFG

SEE ALSO

jbconfig(1), JBCFG(4) in the *CLIX Programmer's & User's Reference Manual.*

NAME

labelit - provide labels for file systems

SYNOPSIS

/etc/labelit *special* [*fsname volume* [**-n**]]

DESCRIPTION

labelit can be used to provide labels for unmounted disk file systems. The **-n** option provides for initial labeling only (this destroys previous contents).

With the optional arguments omitted, *labelit* prints current label values.

The *special* name should be the physical disk section (such as **/dev/dsk/s0u0p7.3**). The device may not be on a remote machine.

The *fsname* argument represents the mounted name (such as **root**, **u1**, etc.) of the file system.

Volume may be used to equate an internal name with a volume name applied externally to the disk pack, diskette, or tape.

Fsname and *volume* are recorded in the super-block.

labelit prints the block size, number of i-nodes, and number of 512-byte sectors of the file system. For file systems of type *ffsfs*(4), the fragment size is also printed.

SEE ALSO

ffsfs(4) in the *CLIX Programmer's & User's Reference Manual.*
makefsys(1M) in the *UNIX System V System Administrator's Reference Manual.*
fs(4) in the *UNIX System V Programmer's Reference Manual.*

NAME

lockd - NFS network lock daemon

SYNOPSIS

/etc/lockd [**-t** *timeout*] [**-g** *graceperiod*] [**-e**]

DESCRIPTION

lockd processes lock requests sent locally by the kernel or remotely by another lock daemon. *lockd* forwards lock requests for remote data to the server site's lock daemon through the *RPC/XDR*(3R) package. *lockd* then requests the status monitor daemon, *statd*(1M), for monitor service. The reply to the lock request is not sent to the kernel until the status daemon and the server site's lock daemon have replied.

If either the status monitor or server site's lock daemon is unavailable, the reply to a lock request for remote data is delayed until all daemons become available.

When a server recovers, it waits for a grace period for all client site *lockd*s to submit reclaim requests. Client site *lockd*s, on the other hand, are notified by the *statd*(1M) of the server recovery and promptly resubmit previously granted lock requests. If a *lockd* fails to secure a previously granted lock at the server site, the *lockd* sends SIGUSR2 to a process.

lockd should be invoked early during the transition from single user to multiuser, so that no other processes can get a standard System V lock (see **-e** option below).

The following options are available:

-t *timeout* — *lockd* uses *timeout* (seconds) as the interval instead of the default value (15 seconds) to retransmit lock request to the remote server.

-g *graceperiod* — *lockd* uses *graceperiod* (seconds) as the grace period duration instead of the default value (45 seconds).

-e — If active locks are in the standard system record-locking code, *lockd* will log a warning to the console. If the **-e** option is specified, *lockd* will exit immediately if there are active standard locks so that the administrator has the option of not effectively destroying a process's locks in progress.

SEE ALSO

statd(1M).
fcntl(2), signal(2) in the *CLIX Programmer's & User's Reference Manual.*

NAME

lpc - BSD line printer control program

SYNOPSIS

/etc/lpc [*command* [*argument* ...]]

DESCRIPTION

lpc is used by the system administrator to control the operation of the line printer system. For each line printer configured in **/etc/printcap**, *lpc* may be used to perform the following:

Disable or enable a printer.

Disable or enable a printer's spooling queue.

Rearrange the order of jobs in a spooling queue.

Find the status of printers, and their associated spooling queues and printer daemons.

Without any arguments, *lpc* will prompt for commands from the standard input. If arguments are supplied, *lpc* interprets the first argument as a command and the remaining arguments as parameters to the command. The standard input may be redirected causing *lpc* to read commands from a file. Commands may be abbreviated. The following is the list of recognized commands.

? [*command* ...]
help [*command* ...]
: Print a short description of each command specified in the argument list, or, if no arguments are given, a list of the recognized commands.

abort {**all** | *printer* ...}
: Immediately terminate an active spooling daemon on the local host and then disable printing (preventing new daemons from being started by *lpr*(1)) for the specified *printer*s.

clean {**all** | *printer* ...}
: Remove any temporary files, data files, and control files that cannot be printed (i.e., do not form a complete printer job) from the specified *printer* queue(s) on the local machine.

disable {**all** | *printer* ...}
: Turn the specified *printer* queues off. This prevents new printer jobs from being entered into the queue by *lpr*(1).

down {**all** | *printer*} *message* ...
: Turn the specified *printer* queue off, disable printing and put *message* in the printer status file. *Message* does not need to be quoted since the remaining arguments are treated like *echo*(1). This is normally used to take a printer down and let others know why. (*lpq*(1) will indicate the *printer* is down and print the status *message*.)

enable {**all** | *printer ...*}
Enable spooling on the local queue for the listed *printers*. This will allow *lpr*(1) to put new jobs in the spool queue.

exit
quit Exit from *lpc*.

restart {**all** | *printer ...*}
Attempt to start a new *printer* daemon. This is useful when some abnormal condition causes the daemon to die unexpectedly leaving jobs in the queue. *lpq*(1) will report that there is no daemon present when this condition occurs. If the user is the super-user, try to abort the current daemon first (i.e., kill and restart a stuck daemon).

start {**all** | *printer ...*}
Enable printing and start a spooling daemon for the listed *printers*.

status Display the status of daemons and queues on the local machine.

stop {**all** | *printer ...*}
Stop a spooling daemon after the current job completes and disable printing.

topq *printer* [*jobnum ...*] [*user ...*]
Place the jobs in the order listed at the top of the *printer* queue.

up {**all** | *printer ...*}
Enable everything and start a new *printer* daemon. Undoes the effects of **down**.

FILES

/etc/printcap	printer description file
/usr/spool/*	spool directories
/usr/spool/*/lock	lock file for queue control

SEE ALSO

lpd(1M).
lpr(1), lpq(1), lprm(1), printcap(4) in the *CLIX Programmer's & User's Reference Manual*.
"BSD LP Spooler Tutorial" in the *CLIX System Guide*.

DIAGNOSTICS

?Ambiguous command
Abbreviation matches more than one command.

?Invalid command
No match was found.

?Privileged command
Command can be executed by root only.

NAME

lpd - BSD line printer daemon

SYNOPSIS

/usr/lib/lpd [-l] [*port#*]

DESCRIPTION

lpd is the line printer daemon (spool area handler) and is normally invoked at boot time from the **rc2.d** file. It makes a single pass through the *printcap*(4) file to find out about the existing printers and prints any files remaining after a crash. It then uses the system calls *listen*(2B) and *accept*(2B) to receive requests to print files in the queue, transfer files to the spooling area, display the queue, or remove jobs from the queue. In each case, it forks a child to handle the request so the parent can continue to listen for more requests. The Internet port number used to rendezvous with other processes is normally obtained with *getservbyname*(3B), but can be changed with the *port#* argument. The **-l** flag causes *lpd* to log valid requests received from the network. This can be useful for debugging.

Access control is provided by two means. First, all requests must come from one of the machines listed in the file **/etc/hosts.equiv** or **/etc/hosts.lpd**. Second, if the "rs" capability is specified in the *printcap*(4) entry for the printer being accessed, *lpr*(1) requests will only be honored for users with accounts on the printer's host.

The file **minfree** in each spool directory contains the number of disk blocks to leave free so that the line printer queue will not completely fill the disk. The **minfree** file can be edited with a text editor.

The file **lock** in each spool directory is used to prevent multiple daemons from becoming active simultaneously and to store information about the daemon process for *lpr*(1), *lpq*(1), and *lprm*(1). After the daemon has successfully set the lock, it scans the directory for files matching the pattern **cf***. Lines in each **cf*** file specify files to be printed or nonprinting actions to be performed. Each such line begins with a key character to specify what to do with the remainder of the line.

J Job Name. String to be used for the job name on the burst page.

C Classification. String to be used for the classification line on the burst page.

L Literal. The line contains identification information from the password file and causes the banner page to be printed.

T Title. String to be used as the title for *pr*(1).

H Host Name. Name of the machine where *lpr*(1) was invoked.

P Person. Login name of the person who invoked *lpr*(1). This is used to verify ownership by *lprm*(1).

M Send mail to the specified user when the current print job completes.

- **f** Formatted File. Name of a file that is already formatted to print.
- **l** Like **f** but passes control characters and does not make page breaks.
- **p** Name of a file to print using *pr*(1) as a filter.
- **t** *Troff* file. The file contains *troff* output.
- **n** *Ditroff* file. The file contains device-independent *troff* output.
- **g** Graph file. The file contains data produced by *plot*(3X).
- **v** The file contains a raster image.
- **r** The file contains text data with FORTRAN carriage control characters.
- **1** *Troff* font R. Name of the font file to use instead of the default.
- **2** *Troff* font I. Name of the font file to use instead of the default.
- **3** *Troff* font B. Name of the font file to use instead of the default.
- **4** *Troff* font S. Name of the font file to use instead of the default.
- **W** Width. Changes the page width (in characters) used by *pr*(1) and the text filters.
- **I** Indent. The number of characters to indent the output by (in ASCII).
- **U** Unlink. Name of file to remove when printing is complete.
- **N** File name. The name of the file which is being printed, or a blank for the standard input (when *lpr*(1) is invoked in a pipeline).

If a file cannot be opened, a message will be logged through the console. *lpd* will try up to 20 times to reopen a file it expects to be there. After attempting to reopen the file, it will skip the file to be printed.

lpd uses *fcntl*(2) to provide exclusive access to the lock file and to prevent multiple daemons from becoming active simultaneously. If the daemon should be killed or die unexpectedly, the lock file need not be removed. The lock file is kept in a readable ASCII form and contains two lines. The first is the process ID of the daemon and the second is the control file name of the current job being printed. The second line is updated to reflect the current status of *lpd* for the programs *lpq*(1) and *lprm*(1).

FILES

/etc/printcap	printer description file
/usr/spool/*	spool directories
/usr/spool/*/minfree	minimum free space to leave
/dev/lp*	line printer devices
/dev/printer	socket for local requests
/etc/hosts.equiv	lists machine names allowed printer access
/etc/hosts.lpd	lists machine names allowed printer access, but not under the same administrative control

SEE ALSO

lpc(1M).
lpr(1), lpq(1), lprm(1), printcap(4) in the *CLIX Programmer's & User's Reference Manual.*
"BSD LP Spooler Tutorial" in the *CLIX System Guide.*

NAME

makedbm - make a YP *dbm* file

SYNOPSIS

/etc/yp/makedbm [**-i** *yp-input-file*] [**-o** *yp-output-name*] [**-d** *yp-domain-name*] [**-m** *yp-master-name*] *infile outfile*

/etc/yp/makedbm [**-u** *dbm-file-name*]

DESCRIPTION

makedbm converts *infile* to a pair of files in *ndbm*(3B) format, namely *outfile*.**pag** and *outfile*.**dir**. Each line of the input file is converted to a single *dbm* record. All characters up to the first tab or space form the key, and the rest of the line is the data. If a line ends with "\", the data for that record is continued to the next line. The clients of the Yellow Pages (YP) will interpret "#"; *makedbm* does not treat it as a comment character. *Infile* can be -, in which case standard input is read.

makedbm is used in generating *dbm* files for the YP and it generates a special entry with the key *yp-last-modified*, which is the date of *infile* (or the current time, if *infile* is -).

The following options are recognized by *makedbm*:

-i *yp-input-file*	Create a special entry with the key *yp-input-file*.
-o *yp-output-name*	Create a special entry with the key *yp-output-name*.
-d *yp-domain-name*	Create a special entry with the key *yp-domain-name*.
-m *yp-master-name*	Create a special entry with the key *yp-master-name*. If no master host name is specified, *yp-master-name* is set to the local host name.
-u *dbmfilename*	Undo a *dbm* file. That is, print out a *dbm* file one entry per line, with a single space separating keys from values.

EXAMPLES

It is easy to write shell scripts to convert standard files such as **/etc/passwd** to the key value form used by *makedbm*. For example, the *awk*(1) program

```
BEGIN { FS = ":"; OFS = "\t"; }
{ print $1, $0 }
```

converts the **/etc/passwd** file to a form that can be read by *makedbm* to make the YP file **passwd.byname**. That is, the key is a user name and the value is the remaining line in the **/etc/passwd** file.

SEE ALSO

yppasswd(1) in the *CLIX Programmer's & User's Reference Manual.*

NAME

makenode - deliverable software installation utility

SYNOPSIS

makenode [**-tfplvmPV**] [**-b** *basedir*] [**-n** *connstr*] [**-F** *prodlist*] [**-T** *tapedev*] [*selection* ...]

DESCRIPTION

makenode installs deliverable Intergraph software products from Intergraph software delivery media to an Intergraph workstation or server. Products loaded to a workstation or server with *makenode* can then be delivered in executable form to workstations or servers using *newprod*(1M). *makenode* can be used interactively or through the command line.

When used interactively, *makenode* initially places user in multi-menu mode. The initial menu lists product classes. After selecting a class, *makenode* displays a menu of available products in the selected class.

By default, products downloaded through *makenode* are placed in **/usr/ws_s** for C100 CLIPPER products and **/usr/ws_ad** for C300 CLIPPER products.

The following are commands available in interactive mode:

u Update selected products.

p Preview fixes of selected products.

m Change menu mode. The **m** option toggles between multi-menu mode and single-screen mode. Single-screen mode allows all available products to be displayed, regardless of class.

q Quit *newprod*.

v Toggle verbose mode of *newprod*. Default is off.

! Execute a shell command.

f Choose file system to download to.

? Get help for interactive options and moving the cursor.

c Clear selected products.

a Automatically select all out-of-date products and clear all previously selected products.

h Help for downloading individual products.

e exit

<BACKSPACE>
<DELETE>
In multi-menu mode, return to previous menu.

<SPACE>
Toggle selection of current product.

k
<UP-ARROW>
<CONTROL>-P
Move cursor up a line.

j
<CONTROL>-N
<DOWN-ARROW>
Move cursor down a line.

<ESC>-v
<CONTROL>-U
<CONTROL>-Z
<LEFT-ARROW>
Move back a page.

<CONTROL>-D
<CONTROL>-V
<RIGHT-ARROW>
Move forward a page.

t
<ESC>-<
Move to the top of the list.

b
<ESC>->
Move to the bottom of the list.

/product
Search for the *product*.

<CONTROL>-L
Refresh the screen.

In command line mode, product numbers are specified on the command line.

Invoked with no options, *newprod* prompts for any required information. *newprod* determines software products available for installation based on the contents of the product list file. The product list file contains a series of records, one per installable product. Each record uses the following format:

number#title#date##size#srcdir#destdir#priority#name#version

Number is the Intergraph-assigned software license number of the product. *Title* is the title of the product. *Date* is the tracking date of the product (see *fixes.com*(4)). *Srcdir* is the location on the Intergraph VAX, workstation, tape, or floppy disk where that product is stored. *Destdir* is the destination directory (appended to the base directory name, see the **-b** option below) where the product will be located on the workstation. *Priority* is the load priority used when loading more than one product at a time. *Name* is the name of the product. *Version* specifies the revision of the product.

The following options are available in *newprod* command line mode:

-b *basedir* — Use the file system given in *basedir* as the base directory for installing software products. A product list file record contains a directory field that is relative to *basedir*. If *basedir* is not specified, the base directory for installation is **/usr**.

-f — Use the floppy disk device **/dev/dsk/floppy** as the source of installation. It is assumed that a floppy disk is inserted in the drive that contains a 2400 (high density) block file system.

-n *connstr* — Use the Ethernet node given in *connstr* as the source of installation. *Connstr* has the form *node*[.*username*[.*password*]], where *node* is an Ethernet Local Area Network (LAN) address or a clearinghouse name for an address; *username* is a valid log-in name on the system whose address is given in node; and *password* is a password for the log-in name supplied in *username*. Only *node* must be supplied when this option is used. *newprod* prompts for a user name if *username* is not entered on the command line. If *username* is terminated by a period, *password* is prompted for with echo disabled.

-p — Pipe the display of the available products menu into *pg*(1). This is useful on line printer terminals. This option becomes the default when the TERM environment variable does not suggest vt100 or vt220.

-v — Display additional error-logging messages. The verbose option also displays the names of files as they are manipulated by *newprod*.

-l — Display the product selections menu in long listing format, providing all the configuration information found in the product list file.

-F *cfg* — Use the file named *cfg* for the product configuration file in place of the default.

-T *tapedev* — Use the device named *tapedev* instead of the default **/dev/rmt/0mn**. Be sure to specify a no rewind device.

-V — Display the version number of *newprod* and then exit.

-P — Preview the **fixes** files of selected products instead of installing them.

FILES

/usr/tmp/ws_?.prod   product list file copied from the installation source

SEE ALSO

newprod(1M), dates(1M).
fixes.com(4), certnote.com(4) in the *CLIX Programmer's & User's Reference Manual*.
pg(1) in the *UNIX System V User's Reference Manual*.

NOTES

Only the super-user may execute this command.

NAME

mkconfig - build a configuration file for a CLIX kernel

SYNOPSIS

mkconfig [**-d** *dir*] [**-u** *alt*] [**-o** *outfile*] [**-m** *asm*] [**-l** *ldlist*] *file* ...

DESCRIPTION

mkconfig concatenates kernel source fragment *file*s to produce a single source file for building CLIX kernels of various configurations. The fragment *file*s may have embedded directives that control the order of concatenation and directives to generate certain configuration data structures for the CLIX kernel.

The following options are available:

-d *dir* Specify the search path to use when reading the fragment *file*s. By default, the current directory is searched.

-u *alt* Append *alt* to the search path and attempt to find the fragment *file*s there first. This allows fragment *file*s common to a family of kernels or machines to be kept in a single directory with unique or machine-dependent information in subdirectories.

-o *outfile* Specify the name of the generated source file. Standard out is the default output file.

-m *asm* Specify the output file for routines generated by the **model** option (see *master*(4)) of the **VECTOR** directive. If the **-m** argument is omitted on the command line, the output of the **model** macro will be assumed to consist of lines of assembly language source code bracketed within C language **asm** directives and included in the primary output. The collective output from all **VECTOR** directives is condensed into a C language data structure named *VECTORS*. The *VECTORS* data structure contains triplets of vector address, interrupt service address, and System Status Word (SSW) values.

-l *ldlist* Specify the output file for the **LOAD** directive. The format of the output file is the format accepted by *mkld*(1M).

mkconfig reads the files in the argument list and copies them to the output file. The output is controlled by directives embedded in the *file*s.

SEE ALSO

mkld(1M), sysconfig(1M).
master(4) in the *CLIX Programmer's & User's Reference Manual.*

WARNINGS

mkconfig does not require the S51K file system to be the first in the file system switch tables as the kernel does. The S51K file system should be created before any others.

If too many named sections are included that also include named sections, *mkconfig* may run out of file descriptors.

NAME

mkfnames - create a full name database for *smail*(1M)

SYNOPSIS

/usr/ip32/sendmail/mkfnames [*file* ...]

DESCRIPTION

mkfnames uses the named *files* as input and writes to standard out a sorted database suitable for use as a full name database for *smail*(1M). The format of an input line is defined by *nptx*(1M). *mkfnames* sends its input to *nptx*(1M), then to *lcasep*(1M), and finally to *sort*(1). If *files* are not specified, **/etc/passwd** is parsed and an attempt is made to convert its contents to the format needed by *nptx*(1M). The correctness of the database generated from **/etc/passwd** cannot be guaranteed because there are a variety of password file formats.

SEE ALSO

lcasep(1M), nptx(1M), smail(1M).
sort(1) in the *UNIX System V User's Reference Manual.*

NAME

mkld - build a link editor file

SYNOPSIS

/usr/src/uts/clipper/build/mkld [-l *basefile*] [*infile*] [*outfile*]

DESCRIPTION

mkld generates a link editor file as output. *Outfile* specifies the name of the output file. Standard out is the default.

Infile specifies the name of the input file. Standard in is the default. The input file may contain an unlimited number of entries. Each entry is separated by a blank line. An entry consists of a unique section name followed by a list of objects that will be loaded into that section. Each object entry must be on a line by itself. The object may be the name of an object file or the name of a library. If only portions of a library are needed, the needed objects from the library are specified on the same line as the library name, separated by spaces or tabs.

The only available option is the following:

-l *basefile* Specify a file that is already in link editor format. This file is used as a base for generating the output file.

The link editor MEMORY directive is created by using all MEMORY directives specified in *basefile* and adding to that one for each entry in *infile*. The names created in the MEMORY directive will be the name of the entry specified in the input file with **_MEM** appended to it. The length is calculated from the size of the text, data, and bss sections of each object listed for the given entry.

Text, data, and bss SECTIONS directives are created for each entry in *infile*. The data and bss will be group aligned to a 4K boundary. Any SECTIONS directives specified in the *basefile* will also be used. Other items in the *basefile* will be copied to the output file unchanged.

EXAMPLES

The following is a sample input file:

```
abc
/usr/src/uts/clipper/build/lib.abc

xyz
        lib.xyz
        /usr/src/uts/clipper/build/lib.io       x.o y.o z.o
```

The following is the output from *mkld* when the above input file is used and the -l option is not used.

```
MEMORY {
        xyz_MEM:    origin = 0x40000000,  length = 0x8000
        abc_MEM:    origin = 0x40008000,  length = 0x18000
}
```

```
SECTIONS {
        xyz_text: {
                ___xyz_INIT = . ;
                TMPDIR/x.o    (.text)
                TMPDIR/y.o    (.text)
                TMPDIR/z.o    (.text)
        } > xyz_MEM

        GROUP ALIGN (0x1000): {
                xyz_data: {
                        TMPDIR/x.o    (.data)
                        TMPDIR/y.o    (.data)
                        TMPDIR/z.o    (.data)
                }
                xyz_bss: {
                        TMPDIR/x.o    [COMMON]
                        TMPDIR/y.o    [COMMON]
                        TMPDIR/z.o    [COMMON]
                }
        } > xyz_MEM

        abc_text: {
                ___abc_INIT = . ;
                lib.abc (.text)
        } > abc_MEM

        GROUP ALIGN (0x1000): {
                abc_data: {
                        lib.abc (.data)
                }
                abc_bss: {
                        lib.abc [COMMON]
                }
        } > abc_MEM
}
```

SEE ALSO

master(4) in the *CLIX Programmer's & User's Reference Manual.*

NAME

mklost+found - make a **lost+found** directory for *fsck*(1M)

SYNOPSIS

/etc/mklost+found

DESCRIPTION

mklost+found creates a number of empty files and then removes them to make empty slots for *fsck*(1M). These files are placed in the directory **lost+found** created by *mklost+found* in the current directory.

SEE ALSO

ffsfsck(1M), ffsmkfs(1M).
fsck(1M), mkfs(1M) in the *UNIX System V System Administrator's Reference Manual.*

NAME

mkpar - create or modify a disk partition table

SYNOPSIS

/etc/mkpar [**-f**] *rawdevice*

DESCRIPTION

mkpar creates or modifies the partition information for any disk device. *mkpar* reads the partition information from standard input and creates the described partitions on *rawdevice*. Each input line describes a partition.

The format of the input line is shown below:

partition<TAB>*modifier*<TAB>*size*

partition The partition number should be in the first column.

modifier The modifier number should be in the second column.

size The partition size should be in the third column.

Any input data beyond the third column is ignored.

As each partition is created the partition information is echoed to standard output. A start block is added to the output in column 4. The start block is calculated from the cumulative sizes of previously specified partitions.

The input format for *mkpar* is the same as the output format of *parck*(1M). This allows for a modified version of the partition table from *parck*(1M) to be used as input to *mkpar*.

The following option is supported:

-f Partition a floppy disk.

SEE ALSO

parck(1M), dc(7S), fl(7S).
diskpar(4), floppypar(4) in the *CLIX Programmer's & User's Reference Manual.*

NAME

mount, umount - mount and unmount file system

SYNOPSIS

/etc/mount
/etc/mount [**-r**] [**-f** *fstype*] *special directory*
/etc/mount [**-r**] **-f NFS**[*,options*] *resource directory*
/etc/mount [**-r**] [**-c**] **-d** *resource directory*
/etc/umount *directory*
/etc/umount -d *resource*

DESCRIPTION

File systems other than root are removable in the sense that they can be available or unavailable to users. *mount* announces to the system that *special*, a block special device, or *resource*, a remote resource, is available to users from the mount point directory. *Directory* must exist; it becomes the name of the root of the newly-mounted *special* or *resource*.

mount, when entered with arguments, adds an entry to the table of mounted devices, **/etc/mnttab**. *umount* removes the entry. If invoked with no arguments, *mount* prints the entire mount table. If invoked with an incomplete argument list, *mount* searches **/etc/fstab** for the missing arguments.

The following options are available:

-r Indicates that *special* or *resource* is to be mounted read-only. If *special* or *resource* is write-protected, this flag must be used.

-d Indicates that *resource* is a remote resource that is to be mounted on *directory* or unmounted. To mount a remote resource, Remote File Sharing must be running and the resource must be advertised by a remote computer (see *rfstart*(1M) and *adv*(1M)).

-c Disables RFS client caching of file system reads and writes on this *resource*.

-f *fstype* Indicates that *fstype* is the file system type to be mounted. If this argument is omitted, it defaults to the root *fstype*. If *fstype* is Network File System (NFS, NFS options may be added after the *fstype* separated by commas. The available NFS options are as follows:

soft Return error if the server does not respond.

rsize=*n* Set the read buffer size to *n* bytes.

wsize=*n* Set the write buffer size to *n* bytes.

timeo=*n* Set the initial NFS timeout to *n* tenths of a second.

retrans=*n* Set the number of NFS retransmissions to *n*.

port=*n* Set the server IP port number to *n*.

special Indicates the block special device to be mounted on *directory*. If *fstype* is NFS, *special* should have the form *host-name:/path-name*.

resource Indicates the remote resource name to be mounted on a directory.

directory Indicates the directory mount point for *special* or *resource*. (The directory must exist.)

umount announces to the system that the file system previously mounted *special* or *resource* is to be unavailable. If invoked with an incomplete argument list, *umount* searches **/etc/fstab** for the missing arguments. *mount* can be used by any user to list mounted file systems and resources.

FILES

/etc/mnttab mount table
/etc/fstab file system table

SEE ALSO

mountd(1M), nfsd(1M), showmount(1M).
mount(2), umount(2), fstab(4), mnttab(4) in the *CLIX Programmer's & User's Reference Manual*.
adv(1M), fuser(1M), rfstart(1M), setmnt(1M), unadv(1M) in the *UNIX System Administrator's Reference Manual*.
"Remote File Sharing" chapter, *UNIX System Administrator's Guide* for guidelines when mounting remote resources.

DIAGNOSTICS

If the *mount*(2) system call fails, *mount* prints an appropriate diagnostic. *mount* issues a warning if the file system to be mounted is currently mounted under another name. A remote resource mount will fail if the resource is not available or if Remote File Sharing is not running.

umount fails if *special* or *resource* is not mounted or if it is busy. *Special* or *resource* is busy if it contains an open file or some user's working directory. In such a case, *fuser*(1M) can be used to list and kill processes using *special* or *resource*.

NOTES

Only the super-user may execute these commands.

WARNINGS

Physically removing a mounted file system floppy disk from the disk drive before issuing the *umount* command damages the file system.

NAME

mountd - NFS mount request server

SYNOPSIS

/etc/mountd

DESCRIPTION

mountd is a Remote Procedure Call (RPC) server that answers file system mount requests. *mountd* reads the file **/etc/exports**, described in *exports*(4), to determine which file systems are available to what machines and users. It also provides information about which clients have file systems mounted. This information can be printed using the *showmount*(1M) command.

SEE ALSO

showmount(1M).
exports(4) in the *CLIX Programmer's & User's Reference Guide.*

NOTES

Only the super-user may execute this command.

NAME

namex - updates hosts file and Intergraph clearinghouse database

SYNOPSIS

/usr/ip32/inc/namex [**-v**] [**-u**] [**-h** *hostsfile*] [**-d** *delay*]

DESCRIPTION

namex is run at system startup, at a time specified in the root crontab entry, and when a product delivery (*newprod*(1M)) sets or changes the Internet address of the system.

namex reads the hosts file (**/etc/hosts** by default) and the clearinghouse database and collates the two. *namex* adds any additional clearinghouse definitions for Internet addresses to the hosts file and creates local clearinghouse objects for additional hosts file entries.

The following options are available:

- **-v** Give informational messages on **stderr**.
- **-u** Tell *namex* to run unconditionally.
- **-h** *hostfile* Specify an alternate hosts file.
- **-d** *delay* Specify a delay (in seconds) before processing starts.

namex delimits the changed area of the hosts file with comments. Added or changed entries in the hosts file should not be entered in the area delimited by comments because they will be deleted the next time *namex* runs. *namex* marks its local clearinghouse objects with properties beginning with the string "namex". These properties are reserved by Intergraph. The objects should not be edited, because they will be rebuilt by *namex* when it runs.

namex never overrides a hosts file or clearinghouse definition. The **-v** option will issue warnings about conflicts. Comments in the hosts file will also note conflicts in the definitions.

namex creates a timestamp file, **/usr/lib/nodes/namex.stamp**. Each time *namex* runs, it checks to see if the clearinghouse or the hosts file is newer than the timestamp. If the timestamp is newer, no processing occurs. If the **-d** option is specified, this check is run before the delay is taken. The **-u** option to *namex* overrides this feature. The **namex.stamp** file is also used by *namex* as a lock so that only one copy of *namex* runs at a time.

FILES

/etc/hosts
/usr/lib/nodes/namex.stamp

NOTES

Read and write permissions are required to access the hosts file, the clearinghouse database, and the timestamp file.

NAME

netmap - provides a map of the local area network

SYNOPSIS

netmap [-?] [*-options*]

DESCRIPTION

netmap is an extension of the Intergraph Network Core Monitor, *incmon*(1M), that provides a "map" of the network. Because of graphics requirements, *netmap* is available only on Intergraph graphic workstations.

Help is available within *netmap*, and is accessed by selecting the Help option with the mouse. For help on command-line options, use the following command:

```
netmap -?
```

FILES

/usr/ip32/resrc/help_proc	online help
/usr/ip32/inc/ntmap/netmap.hlp	online help
/usr/ip32/inc/ntmap/netmap.sym	symbol file
/usr/ip32/resrc/bsfont/dutch801.24	font file
/usr/ip32/resrc/bsfont/dutch801[bi].24	font files
/usr/ip32/resrc/bsfont/mono821[bi].24	font files

SEE ALSO

incmon(1M) in the *CLIX User's Reference Manual.*
Intergraph Network Core User's Guide.

NAME

netserver - DNP universal server

SYNOPSIS

netserver

DESCRIPTION

netserver handles Digital Network Protocol (DNP) connect requests for all of the DECnet objects declared in **/usr/lib/servers.reg**. This is more efficient than having multiple servers running simultaneously. When the *netserver* receives a connect, it invokes a server to process the connect and then waits for the next connect. For each connection, *netserver* performs the following steps.

1. *netserver* checks to see if a user name and password, commonly referred to as Access Control Information (ACI), are specified in the connect request (see *netcp*(1)).

   If the ACI specified is invalid according to the **/etc/passwd** file, the link is rejected.

   netserver then checks the **/usr/lib/servers.priv** file to determine the default privileged and nonprivileged accounts. The file has the following format:

   priv *privileged-account-name*
   null *nonprivileged-account-name*

   The order of the lines in the file do not matter. For example, the **/usr/lib/servers.priv** file may contain the following lines:

   priv root
   null guest

   making "root" the *privileged-account-name* and "guest" the *nonprivileged-account-name*.

   If the ACI is valid and the account has root privilege, *netserver* checks to ensure that the user name agrees with that of the DECnet privileged account listed in the **/usr/lib/servers.priv** file. If the file does not exist, **decnet** is used as the default privileged account. If the user names disagree, *netserver* rejects the link.

   If ACI is not provided, *netserver* uses the nonprivileged account listed in **/usr/lib/servers.priv** or **comunity** if the file does not exist.

   Once an account has been determined, the *netserver* child sets itself up with the privileges, permissions, and default directory of that account. It also opens a **servers.log** log file and appends the connect information to it. *netserver* sets up the **stderr** file descriptor to write to the log file.

2. *netserver* checks the **/usr/lib/servers.reg** file for an entry for the requested object. When an object is requested, it will be referred to

by number or by name, but not both. The requested object may refer to any valid object by name, or request an object number in the 1-255 range. The **/usr/lib/servers.reg** file contains a list of registered servers listed one per line. The format of each line is as follows:

obj-number obj-name path

obj-number is in the 0-255 range. If *obj-number* is 0, the server referred to on that line can be referenced only by *obj-name*, since a client cannot request object number 0. This allows several servers to have a 0 *obj-number*, which can be referenced only by their respective *obj-names*. *Path* is the server to execute when server is requested.

netserver checks for an object name or object number in **/usr/lib/servers.reg**. The following is an example of a standard **servers.reg** file:

0	MYTEST	/usr/tmp/mytest
0	HISTEST	/usr/tmp/histest
17	FAL	/usr/bin/fal
19	NML	/usr/bin/nml
27	MAIL	/usr/bin/cumaild
42	CTERM	/usr/bin/sethostd

If no entry is found and the connection was by name, *netserver* searches the default directory of the user specified for a runnable file named *object-name*.

3. *netserver* starts the process specified by the path found in **/usr/lib/servers.reg** or in the user's login directory with a single argument. That argument is the file descriptor number of the logical link. If the invocation fails, the link is rejected.

FILES

/usr/lib/servers.reg	registered server list
/usr/lib/servers.priv	DNP default accounts

SEE ALSO

fal(1M), nml(1M), cumaild(1M), sethostd(1M).
netcp(1) in the *CLIX Programmer's & User's Reference Manual.*
Digital Network Protocol (DNP) Network Manager's Guide.
Digital Network Protocol (DNP) User's Guide.

NAME

newfs - construct a new file system

SYNOPSIS

/etc/newfs [-**N**] [-**v**] [*option* ...] *special disk-type*

DESCRIPTION

newfs is a "friendly" front-end to *ffsmkfs*(1M). *newfs* looks up the type of disk a file system is being created on in the disk description file **/etc/disktab**, calculates the appropriate parameters to use in calling *ffsmkfs*(1M), and then builds the file system by forking *ffsmkfs*(1M). *newfs* determines the size of the file system to be made by looking at the size of disk partition *special*. The **-N** option causes the file system parameters to be printed without actually creating the file system.

If the **-v** option is supplied, *newfs* prints its actions, including the parameters passed to *ffsmkfs*(1M).

Options that may be used to override default parameters passed to *ffsmkfs*(1M) for the file system are as follows:

-s *size* — The size in sectors. The default *size* will be the size of the partition indicated by special.

-b *block-size* — The block size in bytes.

-f *frag-size* — The fragment size in bytes.

-t *#tracks/cylinder* — The number of tracks per cylinder.

-c *#cylinders/group* — The number of cylinders per cylinder group. The default value is 16.

-m *%minfree* — The percentage of space reserved from normal users; the minimum free space threshold. The default is 10 percent.

-o [**space** | **time**] — The file system can either be instructed to minimize the time spent allocating blocks or to minimize the space fragmentation on the disk. If the value of *minfree* (see above) is less than 10 percent, the default is to optimize for space; if the value of *minfree* is greater than or equal to 10 percent, the default is to optimize for time.

-r *revolutions/minute* — The speed of the disk in revolutions per minute (normally 3600).

-S *sector-size* — The size of a sector in bytes (almost always 512).

-i *bytes-per-i-node* — The density of i-nodes. The default is to create an i-node for each 2048 bytes of data space. If fewer i-nodes are desired, a larger number should be used; to create more i-nodes, a smaller number should be given. This value will be bounded so that the number of i-nodes is less than 65536.

FILES

/etc/disktab disk geometry and file system partition information

SEE ALSO

ffsfsck(1M), ffsmkfs(1M).
disktab(4), ffsfs(4) in the *CLIX Programmer's & User's Reference Manual.*
"FFS Tutorial" in the *CLIX System Guide.*

BUGS

newfs should derive the type of the disk without the user's help.

NAME

newprod, makenode - CLIX software installation utilities

SYNOPSIS

newprod [**-trfcplvxydmRV**] [**-b** *basedir*] [**-n** *connstr*] [**-F** *prodlist*] [**-T** *tapedev*] [*selection* ...]

makenode [**-trfcplvmRV**] [**-b** *basedir*] [**-n** *connstr*] [**-F** *prodlist*] [**-T** *tapedev*] [*selection* ...]

DESCRIPTION

newprod installs software products from software delivery media to an Intergraph workstation or server. *makenode* creates a delivery source on an Intergraph workstation or server. Products loaded on a workstation or server with *makenode* can then be delivered in executable form to workstations or servers with *newprod*. By default, products downloaded using *makenode* are placed in **/usr/ws_s**.

newprod and *makenode* can be used interactively or through the command line.

The following options are available in command-line mode:

-b *basedir* Use the file system given in *basedir* as the base directory for installing software products. A product list file record contains a directory field that is relative to *basedir*. If *basedir* is not specified, the base directory for installation is **/usr**.

-f Use the floppy disk device **/dev/dsk/floppy** as the source of installation. It is assumed that a floppy disk is inserted in the drive that contains a 2400-block (high-density) file system.

-n *connstr* Use the Ethernet node given in *connstr* as the source of installation. *Connstr* has the form *node*[*.username*[*.password*]], where *node* is an Ethernet Local Area Network (LAN) address or a clearinghouse name for an address; *username* is a valid login name on the system whose address is given in *node*; and *password* is a password for the login name supplied in *username*. Only *node* must be supplied when this option is used. *newprod* or *makenode* prompts for a user name if *username* is not entered on the command line. If *username* is terminated by a period, *password* is prompted for with echo disabled.

-p Pipe the display of the available products menu into *pg*(1). This is useful on line printer terminals. This option becomes the default when the **TERM** environment variable does not suggest vt100 or vt220.

-v Display additional error-logging messages. The verbose option also displays the names of files as they are manipulated by *newprod* or *makenode*.

-y
-d Select default answers to any questions presented by the product installation shell scripts.

-x Display the commands executed by installation shell scripts.

-l Display the product selections menu in long listing format, providing all the configuration information found in the product list file.

-F *cfg* Use the file named *cfg* for the product list file instead of the default (which is **ws_s.prods**).

-T *tapedev* Use the device named *tapedev* instead of the default **/dev/rmt/0mn**. A no-rewind device must be specified.

-V Display the version number and then exit.

-R Review the release notes of selected products instead of installing them.

-i Specify that the install scripts will download their Installation Test Procedure (ITP) files.

-m Turn off multimenu mode.

-t Use a tape drive as the source of installation.

-r *connstr* Use a remote CDROM at *connstr* as the source of installation.

-c Use a local CDROM as the source of installation.

In command-line mode, product numbers are specified on the command line.

When used interactively, *newprod* and *makenode* initially place users in multimenu mode. The initial menu lists product classes. After the user selects a class, *newprod* or *makenode* displays a menu of available products in the selected class.

The following commands are available in interactive mode:

u Update selected products.

r Review release notes of selected products.

m Change menu mode. The **m** option toggles between multimenu mode and single-screen mode. Single-screen mode allows all available products to be displayed, regardless of their class.

q Terminate the session.

v Toggle verbose mode. The default is off.

y
d Toggle acceptance of all installation defaults. The default is off.

x Toggle install script debug mode. The default is off.

! Execute a shell command.

f Choose the file system to download to.

? Get help for interactive options and cursor movement.

c Clear selected products.

a Automatically select all out-of-date products and clear all previously selected products.

h Access help for downloading individual products.

e Exit.

<BACK SPACE>
<DELETE>
In multimenu mode, return to the previous menu.

<SPACE>
Toggle selection of current product.

k
<UP-ARROW>
<CONTROL>-P
Move the cursor up a line.

j
<CONTROL>-N
<DOWN-ARROW>
Move the cursor down a line.

<ESC>-v
<CONTROL>-U
<CONTROL>-Z
<LEFT-ARROW>
Move back a page.

<CONTROL>-D
<CONTROL>-V
<RIGHT-ARROW>
Move forward a page.

t
<ESC>-<
Move to the top of the list.

b
<ESC>->
Move to the bottom of the list.

/product
Search for *product*.

<CONTROL>-L
Refresh the screen.

Invoked with no options, *newprod* and *makenode* prompt for any required information. *newprod* and *makenode* determine software products available for installation based on the contents of the product list file. The product list file contains a series of records, one per installable product. Each record uses the following format:

number#title#date##size#srcdir#destdir#priority#name#version#class

Number is the Intergraph-assigned software license number of the product. *Title* is the title of the product. *Date* is the tracking date of the product (see *fixes.com*(4)). *Srcdir* is the location on the Intergraph VAX, workstation, tape, or floppy disk where that product is stored. *Destdir* is the destination directory (appended to the base directory name, see the **-b** option below) where the product will be located on the workstation. *Priority* is the load priority used when loading more than one product at a time. *Name* is the name of the product. *Version* is the revision number of the product. *Class* is the product classification.

FILES

/usr/tmp/ws_s.prod   product list file copied from the installation source

SEE ALSO

dates(1M).
fixes.com(4), certnote.com(4) in the *CLIX Programmer's & User's Reference Manual.*

NOTES

Only the super-user may execute this command.

NAME

nfsd, biod - NFS daemons

SYNOPSIS

/etc/nfsd [*nservers*]
/etc/biod [*nservers*]

DESCRIPTION

nfsd starts the Network File System (NFS) server daemons that handle client file system requests. *Nservers* is the number of file system request daemons to start.

biod, run on an NFS client, starts *nservers* asynchronous block I/O daemons, which read-ahead and write-behind blocks from the client's buffer cache.

SEE ALSO

mountd(1M).
exports(4) in the *CLIX Programmer's & User's Reference Manual.*

NOTES

Only the super-user may execute these commands.

NAME

nfsstat - NFS statistics

SYNOPSIS

/etc/nfsstat [-csnrz]

DESCRIPTION

nfsstat displays statistical information about the Network File System (NFS) and Remote Procedure Call (RPC) interfaces to the kernel. It can also be used to reinitialize this information. If no options are given, the default is *nfsstat* **-csnr**; that is, print everything and reinitialize nothing.

The available options are as follows:

-c Display client information. Only the client side NFS and RPC information is printed. Can be combined with the **-n** and **-r** options to print client NFS or client RPC information only.

-s Display server information. Works like the **-c** option above.

-n Display NFS information. NFS information for both the client and server side is printed. Can be combined with the **-c** and **-s** options to print client or server NFS information only.

-r Display RPC information. Works like the **-n** option above.

-z Zero (reinitialize) statistics. Can be combined with any of the above options to zero particular sets of statistics after printing them. The user must have write permission on **/dev/kmem** for this option to work.

FILES

/unix	system namelist
/dev/kmem	kernel memory

NAME

nml - DNP network management server

SYNOPSIS

nml

DESCRIPTION

nml is the Digital Network Protocol (DNP) server that responds to network management requests from the Network Control Program (NCP). *nml* is invoked by *netserver*(1M).

NCP converts network management commands to Network Information and Control Exchange (NICE) protocol messages and sends them to *nml* for processing. Responses to commands performed by *nml* are returned to *ncp*(1) for display. Network management commands allow the user to monitor the network status, view and set network parameters, and perform other administrative functions.

NOTES

Because *ncp*(1) and *nml* are separate components, it is possible for them to support different sets of functions. To execute network management command, both *ncp*(1) and *nml* must support the command.

SEE ALSO

netserver(1M).
ncp(1) in the *CLIX Programmer's & User's Reference Manual.*
Digital Network Protocol (DNP) Network Manager's Guide.

NAME

nocore - disable/enable core dumping

SYNOPSIS

nocore [-c]

DESCRIPTION

nocore disables or enables core dumping system wide. *nocore* without an argument disables core dumping. With the **-c** option, core dumping is enabled. The system default is core dumping enabled.

NOTES

Only the super-user may execute this command.

NAME

nptx - full name permutations

SYNOPSIS

/usr/ip32/sendmail/nptx

DESCRIPTION

nptx reads a list of address name pairs from standard input and prints name permutations and the address pairs to standard output. *mkfnames*(1M) uses *nptx* to generate a full name database for *smail*(1M). The format of an input line is as follows:

address *name*

The *address* field can contain any address and is terminated by a tab character (ASCII 0x9). This field is not translated. The *name* field consists of names or initials separated by white space with an optional nickname given in parentheses. It is terminated by a newline (ASCII 0xA). All permutations of the names and initials are printed. The only restriction is that the last name will appear in each permutation. The permutations are not necessarily unique.

EXAMPLES

This command will display results as follows:

```
echo "gpb@ECH.gatech.edu\tWrecker Burdell(George P.)"|nptx

Burdell             gpb@ECH.gatech.edu
W.Burdell           gpb@ECH.gatech.edu
Wrecker.Burdell     gpb@ECH.gatech.edu
Burdell             gpb@ECH.gatech.edu
G.Burdell           gpb@ECH.gatech.edu
George.Burdell      gpb@ECH.gatech.edu
P.Burdell           gpb@ECH.gatech.edu
P.Burdell           gpb@ECH.gatech.edu
G.P.Burdell         gpb@ECH.gatech.edu
George.P.Burdell    gpb@ECH.gatech.edu
G.P.Burdell         gpb@ECH.gatech.edu
George.P.Burdell    gpb@ECH.gatech.edu
```

SEE ALSO

mkfnames(1M), smail(1M).

NAME

odfsck - check optical disk file system integrity and correct problems

SYNOPSIS

odfsck [**-n**] [**-R**{**original**|**lostfound**|**rename**|**remove**}
[**-A**{**unique**|**overwrite**|**remove**}]] [*special*] *volume*

DESCRIPTION

odfsck audits and interactively repairs inconsistent conditions for optical disk file systems. If the free block list is corrupt, the last good super-block is found and read. The blocks written after the last good super-block are examined and the super-block is updated to reflect all added or deleted files or directories. The file system is then examined for inconsistencies.

All inconsistencies besides unreferenced files (orphans) are automatically corrected. Orphans are files or directories that cannot be accessed because one or more of the directories in the path specification are missing. The user will be prompted for the correct action for each orphan found unless a default answer for the reconnection options prompt is given.

The following inconsistencies are checked:

1. Free block and/or free i-node count is incorrect.
2. Directory size is not a multiple of the directory entry structure size.
3. Link counts are incorrect.

special is a generic Small Computer System Interface (SCSI) character device specified in the standalone optical disk configuration file. This argument is not specified if the volume is inside a jukebox.

odfsck accepts the following options:

-n Do not write to the file system.

-R Specify the default answer for the reconnection options prompt. The valid values are as follows:

original	Reconnect to the original path specification, creating directories as needed.
lostfound	Reconnect the original path specification's basename to the **::/lost+found** directory.
rename	Reconnect to the name that the user will supply, creating directories as needed.
remove	Clear the i-node that references this file. If the file exists, do not reconnect this orphan.

-A Specify the default answer for the existing options prompt. This prompt is issued if the **-R** option is specified and the path name that the orphan was to be reconnected to exists. Valid values are as follows:

unique Permute the desired name until it is unique. The orphan will then be reconnected to the unique name.

overwrite Use the desired name and overwrite the existing file.

remove Clear the i-node that references this file. If the file exists, do not reconnect this orphan.

SEE ALSO

odintro(1) in the *CLIX Programmer's & User's Reference Manual.*

BUGS

I-node numbers for the . and .. files in each directory are not checked for validity.

NAME

odlabel - create an optical disk file system and label an optical disk volume

SYNOPSIS

odlabel *special volume*

DESCRIPTION

odlabel initializes a volume by labeling it with the name *volume* and initializing the basic file system data structures. It assumes that a platter is in the drive and that the drive has been spun up. Block and i-node counts are not passed to this utility as they are to *mkfs*(1M) because the physical medium determines these parameters.

Special is a generic Small Computer System Interface (SCSI) character device that is specified in the standalone optical disk configuration file **/usr/ip32/od/STANDCFG**. *Volume* is the label name. A volume is one side of an optical disk platter. Volumes are not partitioned. Each volume contains a root directory and a tree of user-defined subdirectories.

FILES

/dev/gs/*
/usr/ip32/od/STANDCFG

SEE ALSO

odmount(1M).
odintro(1), STANDCFG(4) in the *CLIX Programmer's & User's Reference Manual.*

NAME

odmount - mount an optical disk volume

SYNOPSIS

odmount [-r] [*special*] *volume*

DESCRIPTION

For a manual-load drive, *odmount* assumes that a platter is present in the specified drive, *special*, and that the drive has been spun up. This utility verifies the label and reads data structures from the disk into memory for other utilities to use. For a jukebox-resident platter, *odmount* allocates a drive and loads a platter.

The **-r** option mounts *volume* with read-only access.

Special is a generic Small Computer System Interface (SCSI) character device specified in the standalone optical disk configuration file. This argument is not specified if the volume is inside a jukebox.

Volume is the optical disk volume name. A volume is one side of an optical disk platter. Volumes are not partitioned. Each volume contains a root directory and a tree of user-defined subdirectories.

FILES

/dev/gs/*
/usr/ip32/od/STANDCFG

SEE ALSO

odlabel(1M), odumount(1M).
odintro(1), JBCFG(4), STANDCFG(4) in the *CLIX Programmer's & User's Reference Manual*.

NOTES

Only the super-user may execute this command.

WARNINGS

If the volume has been modified and is not unmounted with *odumount*(1M) before the system is shut down, it must be recovered with *odfsck*(1M).

NAME

odreadlabel - read optical disk label information

SYNOPSIS

odreadlabel [**-almcv**] *special*

DESCRIPTION

odreadlabel reads the label information from the optical disk volume currently in the optical disk drive unit connected to the generic Small Computer System Interface (SCSI) character device *special.*

The following options are available:

-a Print all label fields.

-l Print the label name.

-m Print the number of maps per region. These maps are used to find the most current super-block on the optical disk volume.

-c Print the media capacity.

-v Print the file system version number.

If options are not specified, the label name is printed without a header.

FILES

/dev/gs/*
/usr/ip32/od/STANDCFG

SEE ALSO

STANDCFG(4) in the *CLIX Programmer's & User's Reference Manual.*

NAME

odumount - unmount an optical disk volume

SYNOPSIS

odumount *volume*

DESCRIPTION

odumount writes the file system control structures to the optical disk. The utilities can no longer access the volume. If the file system has not been modified, nothing is written to the disk.

Volume is the name of a mounted optical disk volume.

SEE ALSO

odmount(1M).
odintro(1) in the *CLIX Programmer's & User's Reference Manual.*

NOTES

Only the super-user may execute this command.

WARNINGS

If the volume has been modified and is not unmounted before the system is shut down, it must be recovered with *odfsck*(1M).

NAME

parck - read a disk partition table

SYNOPSIS

/etc/parck [**-f**] *rawdevice* [*par mod*]

DESCRIPTION

parck displays the partition information for any disk device. The displayed information includes partition and modifier number as well as size in blocks and start block for each partition.

If *parck* is invoked with *par* and *mod*, it prints only the start block and size for the specified partition. If that partition does not exist, a -1 is returned for both values.

The following option is supported:

-f Examine a partitioned floppy disk.

SEE ALSO

mkpar(1M), dc(7S), fl(7S).
diskpar(4), floppypar(4) in the *CLIX Programmer's & User's Reference Manual.*

NAME

portmap - DARPA port to RPC program number mapper

SYNOPSIS

/etc/portmap

DESCRIPTION

portmap is a server that converts Remote Procedure Call (RPC) program numbers to Defense Advanced Research Project Agency (DARPA) protocol port numbers. It must be running to make RPC calls.

When an RPC server is started, it tells *portmap* the port number it is listening to and the RPC program numbers it is prepared to serve. When a client wishes to make an RPC call to a given program number, it first contacts *portmap* on the server machine to determine the port number where RPC packets should be sent.

SEE ALSO

rpcinfo(1M).

NOTES

Only the super-user may execute this command.

BUGS

If *portmap* crashes, all servers must be restarted.

NAME

qmgr - Network Queuing System (NQS) queue manager

SYNOPSIS

qmgr

DESCRIPTION

qmgr is a program used by the system administrator or system operator to control NQS requests, queues, devices, and general NQS configuration at the local machine.

Definitions

An NQS *request* is a request by a user or user program to perform a function that requires a delay in servicing (after a certain time). Examples of such functions are the scheduling of a shared serial-access resource (such as a printer) and batch job requests. An NQS *manager* identifies someone capable of changing any NQS characteristic on the local machine. An NQS *operator* identifies someone who can execute only the operator commands as a proper subset of all commands provided by the *qmgr* utility.

NQS supports three queue types: *batch*, *device*, and *pipe*.

Batch queues can be used only to execute NQS batch requests. Only NQS batch requests created by the *qsub*(1) command can be placed in a batch queue.

Device queues can be used only to execute NQS device requests. Only NQS device requests created by the *qpr*(1) command can be placed in a device queue.

Pipe queues are used to send NQS requests to other pipe queues or request destination batch or device queues appropriate for the request type. In general, pipe queues combine with network queues to become the mechanism that NQS uses to transport both batch and device requests to queues on remote machines. It is also legal for a pipe queue to transport requests to queues on the same machine.

When a pipe queue is defined, it is given a set of possible destination queues for requests entered in that pipe queue. In this manner, it is possible for a batch or device request to pass through many pipe queues on its way to its ultimate destination, which must eventually be a batch or device queue (matching the request type).

Each pipe queue has an associated *server*. For each request handled by a pipe queue, the associated server that must select a queue destination for the request being handled is spawned. The selection is based on the characteristics of the request and on the characteristics of each queue in the set of destination queues defined for the pipe queue.

Since a different server can be configured for each pipe queue and batch and device queues can have the pipeonly attribute that will admit requests queued only through another pipe queue, respective NQS installations may use pipe queues as a request class mechanism. This places requests that ask for different resource allocations in different queues. Each of these queues can have different associated limits and priorities.

It is also completely possible for a *pipe-client* (pipe queue server) when handling a request to discover that a destination queue will not accept the request for various reasons, including insufficient resource limits to execute the request or a lack of a corresponding account or privilege for queuing at a remote queue. In such circumstances, the request will be deleted and the user will be notified by mail (see *mail*(1)).

NQS has queue access restrictions. For each queue with a queue type other than network, access may be either unrestricted or restricted. If access is unrestricted, any request may enter the queue. If access is restricted, a request can enter the queue only if the requester or the requester's login group has access to that queue. Requests submitted by the super-user are an exception; they are always queued, even if the super-user has not explicitly been given access. *qstat*(1) may be used to determine who has access to a particular queue.

qmgr recognizes commands regardless of uppercase or lowercase characters. Characters shown in bold in the following sections indicate the smallest possible abbreviation for the command.

General Commands

The following commands are available to all users:

exit Exit *qmgr*.

help [*command*]

Get help information. Without an argument, **help** displays information about the available commands. With an argument, **help** displays information about the specified command. The command may be partially specified as long as it is unique. A more complete **help** request yields more detailed information.

The **help** command provides information that is often more extensive than the manual page descriptions.

show all

Display information about the devices, forms, limits, managers, parameters, and queues on the local machine (see the **show** commands below).

show complex [*complex-name*]

Display the status of all NQS queue complexes on this host. If a *complex-name* is specified, output is limited to that queue complex.

show device [*device-name*]

Display the status of all NQS devices on this host. If a *device-name* is specified, output is limited to that device.

show forms

Display the list of valid forms.

show limits_supported

Display the list of meaningful NQS resource limits on this machine. If a limit is meaningful on a machine, the corresponding *qmgr* commands allow the association of a limit with any batch queue on that machine. Note that users may request resource limits that are not meaningful on the machine where *qsub*(1) is invoked. If the request is to be executed on a remote machine where the limit is meaningful, NQS honors it. Otherwise, the unsupported limit is ignored.

show long queue [*queue-name* [*user-name*]]

Display in long format the status of all NQS queues on this host. If a *queue-name* is specified, output will be limited to that queue. If a *user-name* is specified, output will be limited to requests owned by *user-name*.

show managers

Display the list of NQS managers.

show parameters

Display the general NQS parameters.

show queue [*queue-name* [*user-name*]]

Display the status of all NQS queues on this host. If a *queue-name* is specified, output will be limited to that queue. If a *user-name* is specified, output will be limited to requests owned by *user-name*.

Operator Commands

The following commands require the user to have NQS operator privileges:

abort queue *queue* [*seconds*]

Abort all running requests in the *queue*. A SIGTERM signal is sent to each process of each running request. After the specified number of *seconds* of real time have elapsed, a SIGKILL signal is sent to all remaining processes for each of the running requests. The default number of *seconds* is 60. All requests aborted by this command are deleted and all output files associated with the requests are returned to the appropriate destination.

delete request *request-id* ...

Delete the requests named by *request-ids*. This command can delete both running and nonrunning requests. If a request is running, all processes of the request are sent a SIGKILL signal.

disable device *device*

Complete the current request. After that, the *device* is prevented from handling any more requests until it is enabled. If the disabled *device* was the last enabled device in a queue-to-device mapping, the device queue is effectively stopped.

disable queue *queue*

Prevent any more requests from being placed in this *queue*.

enable device *device*
Make the *device* available to handle requests.

enable queue *queue*
Make the *queue* now available to accept new requests.

hold *request* ...
Place a queued or waiting request in the operator hold state. The *request* will remain in the hold state until it is unheld or released by an operator.

lock local_daemon
Lock the NQS local daemon into memory (see *plock*(2)).

modify request *request* **priority** = *n*
Modify the scheduling *priority* of the *request*. If the *request* is running, the new scheduling *priority* will not take effect unless the *request* returns to a waiting state.

move queue *from-queue to-queue*
Move all requests in the queue specified by *from-queue* to the queue specified by *to-queue*.

move request *request-id queue*
Move the request specified by *request-id* from its current queue to the queue specified by *queue*.

purge queue *queue*
Purge all queued requests from the *queue*. The purged queues are irretrievably lost. Running requests in the *queue* are allowed to complete.

release *request* ...
Remove the *request* from any held or waiting state and place it in the queued state ready to run. This command will release any operator or user hold.

restart queue *queue* [*seconds*]
Send all running requests in the queue specified by *queue* a SIGTERM signal. After a grace period of *seconds* has passed, a SIGKILL is sent to the requests. The requests are then immediately rescheduled for execution. If *queue* was put in a stopping state before the restart was performed, the requests remain queued. If *seconds* is not specified, a default value of 20 seconds is used.

set complex run_limit = *run-limit complex*
Change the *run-limit* of an NQS queue *complex*. The *run-limit* determines the maximum number of requests allowed to run in the queue *complex* at any given time.

set run_limit = *run-limit queue*
Change the *run-limit* of an NQS batch or pipe *queue*. The *run-limit* determines the maximum number of requests allowed to run in the *queue* at any given time.

shutdown [*seconds*]
Shut down NQS on the local host. A SIGTERM signal is sent to each process of each running request. After the specified number of *seconds* of real time have elapsed, a SIGKILL signal is sent to all remaining processes for each running request. If a *seconds* value is not specified, the delay is 60 seconds. Unlike the **abort queue** command, **shutdown** requeues all of the requests it kills if the initial SIGTERM signal is caught or ignored by the running request.

start queue *queue*
Start the *queue* and make requests in the *queue* eligible for selection.

stop queue *queue*
Allow any running requests in the *queue* to complete. All other requests in the *queue* are frozen. New requests can still be submitted to the queue but are also frozen.

unhold *request* ...
Remove the *request* from an operator hold state. If the *request* is also in a user hold, the *request* will remain in the hold state. If the *request* is waiting for a certain time, the *request* will go into the waiting state. Otherwise, the *request* returns to the queued state.

unlock local_daemon
Remove a lock that has been retaining the NQS local daemon in memory (see *plock*(2)).

Manager Commands

The following commands require the user to have NQS manager privileges.

add destination = *destination queue*
add destination = (*destination* [, *destination* ...]) *queue*
Add the specified *destinations* as valid destinations for a pipe queue named *queue*.

add device = *device queue*
add device = (*device* [, *device* ...]) *queue*
Add the specified *devices* as resources to service requests from *queue*. The *devices* must exist (see **create device** below).

add forms *form-name* ...
Add the specified *form-names* to the list of valid forms.

add groups = *group queue*
add groups = (*group* [, *group* ...]) *queue*
Add the specified *groups* to the access list for *queue*. There are two ways to specify a *group*:

group-name
[*group-id*]

add queues = *queue complex*
add queues = (*queue* [, *queue* ...]) *complex*
Add the specified *queues* to the queue complex named *complex*.

add managers *manager* ...

Add the specified *managers* to the list of authorized NQS managers with privileges as specified. A *manager* specification consists of an account name specification followed by either **:m** or **:o**. There are four ways to specify an account name:

local-account-name
[*local-user-id*]
[*remote-user-id*]@*remote-machine-name*
[*remote-user-id*]@[*remote-machine-id*]

If the *account-name* specification is followed by **:m**, the account is designated as an NQS manager account capable of using all *qmgr* commands. If the account name specification is followed by **:o**, the account is designated as an NQS operator account capable of using commands only appropriate for an NQS operator.

add users = *user queue*
add users = (*user* [, *user* ...]) *queue*

Add the specified *users* to the access list for *queue*. There are two ways to specify a *user*:

user-name
[*user-id*]

create **batch_queue** *queue* **priority =** *n* [**pipeonly**] [**run_limit =** *n*]

Define a batch queue named *queue* with interqueue priority *n* (0-63). If **pipeonly** is specified, requests may enter this *queue* only if their source is a pipe queue. The specification of a **run_limit** limits the maximum number of requests allowed to run in the batch queue at any given time. The default run limit is 1.

create complex = *queue complex*
create complex = (*queue* [, *queue* ...]) *complex*

Create a queue *complex* consisting of the specified set of batch and device *queues*. Queue complexes provide for the grouping of a set of batch and device queues into a queue complex, which can have an associated run limit.

create **device** *device* **forms =** *forms* **fullname =** *file-name* server = (*server*)

Define a *device* with the specified *forms* and associate it with a *server*. This is done by specifying an absolute path name to the program binary *server* and any arguments required by the program. *file-name* is the absolute path name of the device (special file) and is typically **/dev/device.**

create **device**_queue *queue* **priority =** *n* [device = *device*] [device = (*device* ...)] [pipeonly]

Define a device queue named *queue* with interqueue priority *n* (0-63). If **pipeonly** is specified, requests may enter this *queue* only if their source is a pipe queue. Following **device**, a list of one or more *devices* that may service this *queue* appears.

create pipe_queue *queue* **priority = *n***
server = (*server*) [destination = (*destination* [, *destination* ...]]
[pipeonly] [run_limit = *n*]
Define a pipe queue named *queue* with interqueue priority *n* (0-63) and associate it with a *server*. This is done by specifying an absolute path name to the program binary *server* and any arguments required by the program. Following **destination** is a list of one or more *destination* queues that may receive requests from this pipe *queue*. If **pipeonly** is specified, requests may enter this *queue* only if their source is a pipe queue. **run_limit** limits the maximum number of requests allowed to run in the pipe queue at any given time. The default run limit is 1.

delete complex *complex*
Delete the queue *complex*.

delete destination = *destination queue*
delete destination = (*destination* [, *destination* ...]) *queue*
Delete the mappings from the pipe *queue* to the *destination* queues. All requests from the named *queue* being transferred to a deleted *destination* complete normally. If all *destinations* for a pipe *queue* are deleted in this manner, the pipe *queue* is effectively stopped.

delete device *device*
Delete the specified *device*. A device must be disabled to delete it from the device set (see **disable device** below).

delete device = *device queue*
delete device = (*device* [, *device* ...]) *queue*
Delete the mappings from the device queue *queue* to the *devices*. All requests from the named device *queue* running on any of the named *devices* are allowed to complete normally. If all queue-to-device mappings for the named device *queue* are removed by this command, the *queue* is effectively stopped.

delete forms *form-name* ...
Delete the specified *form-names* from the list of valid forms.

delete groups = *group queue*
delete groups = (*group* [, *group* ...]) *queue*
Delete the specified *groups* from the access list for *queue*. There are two ways to specify a *group*:

group-name
[*group-id*]

delete managers *manager* ...
Delete the specified *managers* from the list of authorized NQS managers. A *manager* specification consists of an account name specification followed by either **:m** or **:o**. There are four ways to specify an account name:

local-account-name
[*local-user-id*]
[*remote-user-id*]@*remote-machine-name*
[*remote-user-id*]@[*remote-machine-id*]

If the account name specification is followed by **:m**, the account is currently permitted to use all *qmgr* commands. If the account name specification is followed by **:o**, the account is currently permitted to use only the commands appropriate for an operator to use. The super-user always has full privileges.

delete queue *queue*
Delete the *queue*. To delete a *queue*, requests may not be in the *queue* and the *queue* must be disabled. Any queue-to-device mappings are updated accordingly.

delete users = *user queue*
delete users = (*user* [**,** *user* ...]**)** *queue*
Delete the specified *users* from the access list for *queue*. There are two ways to specify a *user*:

user-name
[*user-id*]

remove queue = *queue complex*
remove queue = (*queue* [**,** *queue* ...]**)** *complex*
Remove the specified *queues* from the queue complex named *complex*. If the *queue* is the last queue in the queue *complex*, the queue *complex* is deleted.

set debug *level*
Set the debug *level*. The following values are valid:

0 no debug
1 minimum debug
2 maximum debug

set default batch_request priority *priority*
Set the default intra-batch-queue *priority*. This is not the CLIX execution time priority. This is the priority used if the user does not specify an intraqueue priority parameter on the *qsub*(1) command.

set default batch_request queue *queue*
Set the default batch *queue*. This queue is used if the user does not specify a queue parameter on the *qsub*(1) command.

set default destination_retry time *retry-time*
Set the default number of hours that can elapse during which a pipe queue destination can be unreachable before being marked as completely failed.

set default destination_retry wait *interval*
Set the default number of minutes to wait before retrying a pipe queue destination that was unreachable at the last attempt.

set default device_request priority *priority*
Set the default intra-device-queue *priority*. This is the priority used if the user does not specify an intraqueue priority parameter on the *qpr*(1) command.

set default print_request forms *form-name*
Set the default print forms to *form-name*. This is the form used if the user does not specify a forms parameter on the *qpr*(1) command.

set default print_request queue *queue*
Set the default print *queue*. This is the queue used if the user does not specify a queue parameter on the *qpr*(1) command.

set destination = *destination queue*
set destination = (*destination* [, *destination* ...]**)** *queue*
Associate one or more *destination* queues with a particular pipe *queue*.

set device = *device queue*
set device = (*device* [, *device* ...]**)** *queue*
Associate one or more *devices* with a particular *queue*.

set device_server = (*server***)** *device*
Associate a *server* with a *device*. The *server* consists of the absolute path name to the program binary followed by any arguments required by the program.

set forms *form-name* ...
Specify the valid *form-names*. Other valid forms may be added to this list (see **add forms** above).

set forms = *form-name device*
Set the *form-name* for a *device*.

set lifetime *lifetime*
Set pipe queue request *lifetime* in hours.

set log_file *file-name*
Specify the name of the log file for NQS messages.

set mail *user-id*
Specify the *user-id* used to send NQS mail.

set managers *manager* ...
Set the list of authorized NQS managers to the specified *managers*. A *manager* specification consists of an account name specification followed by either **:m** or **:o**. There are four ways to specify an account name:

local-account-name
[*local-user-id*]
[*remote-user-id*]@*remote-machine-name*
[*remote-user-id*]@[*remote-machine-id*]

If the account name specification is followed by **:m**, the account is designated as an NQS manager account capable of using all *qmgr* commands. If the account name specification is followed by **:o**, the account is designated as an NQS operator account capable of using only those commands appropriate for an NQS operator. The super-user always has full privileges.

set maximum copies *copies*
Set the maximum number of print *copies*.

set maximum open_retries *retries*
Specify the maximum number of *retries* for a failed device open.

set maximum print_size *size*
Specify the maximum *size* of an NQS print file in bytes.

set network client = (*client***)**
Specify the network client to be used. The *client* consists of the absolute path name of the client followed by any arguments required by the client.

set network daemon = (*daemon***)**
Specify the network daemon to be used. *Daemon* consists of the absolute path name of the daemon followed by any arguments required by the daemon.

set network server = (*server***)**
Specify the network server to be used. The *server* consists of the absolute path name of the server followed by any arguments required by the server.

set nice_value_limit = *nice-value queue*
Set the limit for the nice value of requests in a batch *queue*. If a request already in the queue has requested treatment more favorable than the new nice value, it will be given a grandfather clause. A request specifying a nice value may enter a batch queue only if the queue's *nice-value* is numerically less than (more willing to allow access to the CPU) or equal to the request's nice value. The *nice-value* is an integer preceded by an optional negative sign.

set no_access *queue*
Specify that no one will be allowed to place requests in the *queue*. The super-user is an exception; requests submitted by the super-user are always allowed in a queue, even if the super-user is not explicitly given access.

set no_default batch_request queue
Indicate that a default batch request queue will not exist.

set no_default print_request forms
Indicate that default print request forms will not exist.

set no_default print_request queue
Indicate that a default print request queue will not exist.

set no_network_daemon
Indicate that a network daemon will not exist.

set open_wait *interval*
Specify the number of seconds to wait between failed device opens.

set per_process permfile_limit = (*limit* **)** *queue*
Set a per-process maximum file size *limit* for a batch *queue* to which the per-process maximum file size limit for a request may be compared. If the local host does not support per-process file size limits, this command will report an error. Otherwise, every batch queue on the local host will have a per-process maximum file size limit associated with it at all times. If a request already in the queue has requested more than the new limit, it will be given a grandfather clause. A request specifying a per-process maximum file size limit may enter a batch queue only if the queue's limit is greater than or equal to the request's limit.

The format for *limit* is either *.fraction*[*units*] or *integer*[*.fraction*][*units*] when the limit is a finite limit. If an infinite limit is needed, the *limit* may be specified as unlimited or any initial substring. The *integer* and *fraction* portions of a finite limit may be specified as strings of up to eight decimal digits. The *units* may be specified as one of the following case-sensitive strings:

b	bytes
w	words
kb	kilobytes (2^10 bytes)
kw	kilowords (2^10 words)
mb	megabytes (2^20 bytes)
mw	megawords (2^20 words)
gb	gigabytes (2^30 bytes)
gw	gigawords (2^30 words)

When *units* are not specified, bytes are assumed. If the limit is set to unlimited, the only limitations imposed are those of the physical hardware involved.

set pipe_client = (*client* **)** *queue*
Associate a pipe *client* with a pipe *queue*. The *client* consists of the absolute path name to the program binary followed by any arguments required by the program.

set priority = *priority queue*
Specify the interqueue *priority* of a *queue*.

set shell_strategy fixed = (*shell* **)**
Specify that *shell* should be used to execute all batch requests. *Shell* must be the absolute path name of a command interpreter.

set shell_strategy free
Specify that the free shell strategy should be used to execute all batch requests. The free shell strategy duplicates the shell choice

that would have been made if the batch request script had been executed interactively. Under this strategy, the user's login shell is allowed to determine the shell to be used to execute the batch request. The user's login shell is the shell named within the user's entry in the password file (see *passwd*(4)).

set shell_strategy login
: Specify that the login shell strategy should be used to execute all batch requests. Under the login shell strategy, the user's login shell is used to execute the batch request. The login shell is the shell named in the password file (see *passwd*(4)).

set unrestricted_access *queue*
: Specify that requests will not be refused from *queue* because of queue access restrictions.

SEE ALSO

qdel(1), qdev(1), qlimit(1), qpr(1), qstat(1), and qsub(1) in the *CLIX Programmer's & User's Reference Manual.*

NAME

rexecd - remote execution server

SYNOPSIS

/usr/ip32/tcpip/rexecd

DESCRIPTION

rexecd is the server for the *rexec*(3B) routine. The server provides remote execution facilities with authentication based on user names and passwords.

rexecd listens for service requests at the port indicated in the "exec" service specification (see *services*(4)). When a service request is received, the following protocol is initiated:

1) The server reads characters from the socket up to a null ("\0") byte. The resulting string is interpreted as an ASCII number, base 10.

2) If the number received in step 1 is nonzero, it is interpreted as the port number of a secondary stream to be used for the **stderr**. A second connection is then created to the specified port on the client's machine.

3) A null-terminated user name of, at most, 16 characters is retrieved on the initial socket.

4) A null-terminated, unencrypted password of at most 16 characters is retrieved on the initial socket.

5) A null-terminated command to be passed to a shell is retrieved on the initial socket. The length of the command is limited by the upper bound on the size of the system's argument list.

6) *rexecd* then validates the user as is done at login time and, if the authentication was successful, changes to the user's home directory and establishes the user and group protections of the user. If any of these steps fail, the connection is aborted with a diagnostic message returned.

7) A null byte is returned on the initial socket and the command line is passed to the normal login shell of the user. The shell inherits the network connections established by *rexecd*.

SEE ALSO

inetd(1M).
rexec(3B), services(4) in the *CLIX Programmer's & User's Reference Manual.*

DIAGNOSTICS

Except for the last one listed below, all diagnostic messages are returned on the initial socket, after which any network connections are closed. An error is indicated by a leading byte with a value of 1. (0 is returned in step 7 above on successful completion of all the steps before the command execution.)

Username too long.
  The name is longer than 16 characters.

Password too long.
  The password is longer than 16 characters.

Command too long.
  The command line passed exceeds the size of the argument list (as configured into the system).

Login incorrect.
  No password file entry for the user name existed.

Password incorrect.
  The wrong password was supplied.

No remote directory.
  The *chdir*(2) command to the home directory failed.

Try again.
  A *fork*(2) by the server failed.

Shellname: ...
  The user's login shell could not be started. This message is returned on the connection associated with the **stderr** and is not preceded by a flag byte.

BUGS

Indicating "Login incorrect" as opposed to "Password incorrect" is a security breach that allows people to probe a system for users with null passwords.

A facility to allow all data and password exchanges to be encrypted should be present.

NAME

rlogind - remote login server

SYNOPSIS

/usr/ip32/tcpip/rlogind

DESCRIPTION

rlogind is the server for *rlogin*(1). The server provides a remote login facility with authentication based on privileged port numbers from trusted hosts.

rlogind listens for service requests at the port indicated in the "login" service specification (see *services*(4)). When a service request is received the following protocol is initiated:

1. The server checks the client's source port. If the port is not in the range 0-1023, the server aborts the connection.
2. The server checks the client's source address and requests the corresponding host name (see *gethostbyaddr*(3B) and *hosts*(4)). If the host name cannot be determined, the dot-notation representation of the host address is used.

Once the source port and address have been checked, *rlogind* allocates a pseudo terminal (see *pty*(7S)) and manipulates file descriptors so that the slave half of the pseudo terminal becomes the **stdin, stdout,** and **stderr** for a login process. The login process is an instance of the *login*(1) program.

The parent of the login process manipulates the master side of the pseudo terminal, operating as an intermediary between the login process and the client instance of the *rlogin*(1) program. In normal operation, the packet protocol described in *pty*(7S) is invoked to provide <CONTROL-S>/<CONTROL-Q> type facilities and propagate interrupt signals to the remote programs. The login process propagates the client terminal's baud rate and terminal type as found in the environment variable TERM (see *environ*(5)).

SEE ALSO

inetd(1M).

DIAGNOSTICS

All diagnostic messages are returned on the connection associated with the **stderr**, after which any network connections are closed. An error is indicated by a leading byte with a value of 1.

Try again.
: A *fork*(2) by the server failed.

/bin/sh: ...
: The user's login shell could not be started.

BUGS

The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is not secure but is useful in an open environment.

A facility to allow all data exchanges to be encrypted should be present.

A more extensible protocol should be used.

NAME

route - manually manipulate the routing tables

SYNOPSIS

/etc/route [**-f**] [**-n**] [*command* [[**net**] *destination gateway* [*metric*]]]

DESCRIPTION

route is used to manually manipulate the network routing tables. It normally is not needed, as the system routing table management daemon, *routed*(1M), should handle this task.

route uses the I_RIADD, I_RIDEL, and the I_RITBL *ioctl*(2). As such, only the super-user may modify the routing tables.

route supports the following options:

-f Flush the routing tables of all gateway entries. If this is used with one of the commands described above, the tables are flushed before the command's application.

-n Suppress attempts to print host and network names symbolically when reporting actions.

route accepts the following *commands*:

add Add a route. When the specified route is added to the table, the following message is displayed:

add: lan: %*s* rtr: %*s* flags: %*x* proto: %*x*

The displayed values come from the routing table entry supplied in the *ioctl*(2) call. If the gateway address used was not the primary gateway address (the first one returned by *gethostbyname*(2B)), the gateway address is printed numerically as well as symbolically.

delete Delete a route. When a route is deleted from the table, the following message is displayed:

delete: lan: %*s* rtr: %*s* flags: %*x* proto: %*x*

The displayed values come from the routing table entry supplied in the *ioctl*(2) call. If the gateway address used was not the primary gateway address (the first one returned by *gethostbyname*(2B)), the gateway address is printed numerically as well as symbolically.

The -f option will also produce delete messages.

If a delete operation was attempted for an entry that was not present in the tables, the following message is displayed:

Entry not found in kernel routing table. Network: %*s* Gateway: %*s*

list List the kernel routing table.

Destination is the destination network, *gateway* is the next-hop gateway to which packets should be addressed, and *metric* is a count indicating the number of hops to the *destination*.

Metric is required for the **add** command. *Metric* must be 0 if the destination is on a directly attached network and nonzero if the route uses one or more gateways.

If adding a route with *metric* 0, the gateway given is the address of this host on the common network, indicating the interface to be used for transmission.

The optional keyword **net** is provided for compatibility and does not effect the operation of the *route* command.

Host routes are not supported.

If the route leads to a destination connected by a gateway (that is, the route is not to a logically equivalent network), *metric* should be greater than 0.

All symbolic names specified for a *destination* are looked up with *getnetbyname*(2B). Symbolic names specified for *gateway* are looked up with *gethostbyname*(2B).

SEE ALSO

routed(1M).
ioctl(2) in the *CLIX Programmer's & User's Reference Manual.*

NAME

routed - network routing daemon

SYNOPSIS

/etc/routed [-d] [-g] [-s] [-q] [-t] [*logfile*]

DESCRIPTION

routed is invoked at boot time to manage the network routing tables. *routed* uses a variant of the Xerox Network System (XNS) Routing Information Protocol (RIP) in maintaining up-to-date kernel routing table entries. It uses a generalized protocol capable of use with multiple address types, but is currently used only for Internet routing within a cluster of networks.

In normal operation, *routed* listens on the *udp*(7B) socket for the route service for routing information packets (see *services*(4)). If the host is an internetwork router, it periodically supplies copies of its routing tables to any directly connected hosts and networks.

When *routed* is started, it uses the I_IFGETCONF *ioctl*(2) to find directly connected interfaces configured into the system and marked up. If multiple interfaces are present, it is assumed that the host will forward packets between networks. *routed* then transmits a request packet on each interface (using a broadcast packet if the interface supports it) and enters a loop, listening for request and response packets from other hosts.

When a request packet is received, *routed* formulates a reply based on the information maintained in its internal tables. The response packet generated contains a list of known routes, each marked with a hop count metric. (A count of 16 or greater is infinite.) The metric associated with each route returned provides a metric relative to the sender.

Response packets received by *routed* are used to update the routing tables if one of the following conditions is satisfied:

- No routing table entry exists for the destination network or host, and the metric indicates the destination is reachable. (The hop count is not infinite.)
- The source host of the packet is the same as the router in the existing routing table entry. That is, updated information is being received from the internetwork router through which packets for the destination are being routed.
- The existing entry in the routing table has not been updated for some time (defined to be 90 seconds) and the route is at least as short as the current route.
- The new route describes a route to the destination shorter than the one currently stored in the routing tables; the metric of the new route is compared to the one stored in the table.

When an update is applied, *routed* records the change in its internal tables and updates the kernel routing table. The change is reflected in the next response packet sent.

In addition to processing incoming packets, *routed* also periodically checks the routing table entries. If an entry has not been updated for three minutes, the entry's metric is set to infinity and marked for deletion. Deletions are delayed an additional 60 seconds to ensure that the invalidation is propagated throughout the local Internet.

Hosts acting as internetwork routers gratuitously supply their routing tables every 30 seconds to all directly connected hosts and networks. The response is sent to the broadcast address on networks capable of that function, to the destination address on point-to-point links, and to the router's own address on other networks. The normal routing tables are bypassed when gratuitous responses are sent. The reception of responses on each network is used to determine that the network and interface are functioning correctly. If no response is received on an interface, another route may be chosen to route around the interface, or the route may be dropped if no alternative is available.

routed supports the following options:

-d Enable additional debugging information to be logged, such as bad packets received.

-g Offer a route to the default destination on internetwork routers. This is typically used on a gateway to the Internet or on a gateway that uses another routing protocol whose routes are not reported to other local routers.

-s Force *routed* to supply routing information whether it is acting as an internetwork router or not. This is the default if multiple network interfaces are present or if a point-to-point link is in use.

-q Cause the opposite effect of the **-s** option.

-t Print all packets sent or received on the standard output. In addition, *routed* will not separate from the controlling terminal so that interrupts from the keyboard will kill the process.

logfile is interpreted as the name of a file in which *routed*'s actions should be logged. This log contains information about any changes to the routing tables and, if not tracing all packets, a history of recent messages sent and received that are related to the changed route.

In addition to the facilities described above, *routed* supports the concept of distant, passive, and active gateways. When *routed* is started, it reads the file **/etc/gateways** to find gateways that may not be located using only information from the I_IFGETCONF *ioctl*(2). Gateways specified in this manner should be marked as passive if they are not expected to exchange routing information, while gateways marked as active should be willing to exchange routing information. (They should have a *routed* process running on the machine.) Passive gateways are maintained in the routing tables forever and information regarding their existence is included in any routing information transmitted. Active gateways are treated equally to network interfaces. Routing information is distributed to the gateway and if no routing information is received for a period of the time, the associated route is deleted. External gateways are also passive, but are not placed in the kernel routing table or included in routing updates. The function of external entries is to inform *routed* that another routing process will install such a route and that alternate routes to that destination should not be installed. Such entries are required only when both routers may learn of routes to the same destination.

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

net *name1* **gateway** *name2* **metric** *value* [**passive**|**active**|**external**]

The **net** keyword indicates that the route is to a network.

Name1 is the name of the destination network or host. This may be a symbolic name located in **/etc/networks** or **/etc/hosts** or an Internet address specified in dot notation (see *inet*(3B)).

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

Value is a metric indicating the hop count to the destination host or network. If *value* is 0, *name1* is a logically equivalent network for the interface with address *name2*.

Either the **passive**, **active**, or **external** keyword indicates whether the gateway should be treated as passive or active (as described above) or if the gateway is external to the scope of the *routed* protocol.

Internetwork routers that are directly attached to the Advanced Research Project Agency Network (ARPANET) or Military Network (MILNET) should use the Exterior Gateway Protocol (EGP) to gather routing information rather than using a static routing table of passive gateways. EGP is required to provide routes for local networks to the rest of the Internet system.

FILES

/etc/gateways distant gateways

SEE ALSO

udp(7S), gated(1M).

CAVEATS

The kernel's routing tables may not correspond to those of *routed* when redirects change or add routes.

NAME

rpcinfo - report RPC information

SYNOPSIS

/etc/rpcinfo -p [*host*]
/etc/rpcinfo -u *host program-number* [*version-number*]
/etc/rpcinfo -t *host program-number* [*version-number*]

DESCRIPTION

rpcinfo makes a Remote Procedure Call (RPC) to an RPC server and reports what it finds. The following options are available:

-p Probe the portmapper on *host* and print a list of all registered RPC programs. If *host* is not specified, it defaults to the node name returned by *uname*(1).

-u Make an RPC call to procedure 0 of *program-number* using the User Datagram Protocol (UDP) and report whether a response was received.

-t Make an RPC call to procedure 0 of *program-number* using the Transmission Control Protocol (TCP) and report whether a response was received.

The *program-number* argument can be either a name or a number. If no *version-number* is given, it defaults to 1.

FILES

/etc/rpc names for RPC program numbers

SEE ALSO

portmap(1M).
rpc(4) in the *CLIX Programmer's & User's Reference Manual*.
"RPC/XDR Tutorial" in the *CLIX System Guide*.

NAME

rpipe_s - remote pipe server

SYNOPSIS

/usr/ipc/inc/rpipe_s

DESCRIPTION

rpipe_s is a server for *rpipe*(1), started by the *xns_listener*(1M). *rpipe_s* starts a shell with the -c option, and passes it the *command-list* specified by *rpipe*(1). All data redirected to *rpipe*(1) is redirected to *command-list*.

SEE ALSO

xns_listener(1M).
rpipe(1), server.dat(4) in the *CLIX Programmer's & User's Reference Manual.*

WARNINGS

This program can only be started by the *xns_listener*(1M).

NAME

rshd - remote shell server

SYNOPSIS

/usr/ip32/tcpip/rshd

DESCRIPTION

rshd is the server for the *rcmd*(3B) routine and, consequently, for the *rcmd*(1) program. The server provides remote execution facilities with authentication based on privileged port numbers from trusted hosts.

rshd listens for service requests at the port indicated in the *cmd* service specification (see *services*(4)). When a service request is received, the following protocol is initiated:

1. The server checks the client's source port. If the port is not in the 0-1023 range, the server aborts the connection.

2. The server reads characters from the socket up to a null (\0) byte. The resulting string is interpreted as an ASCII number, base 10.

3. If the number received in step 2 is nonzero, it is interpreted as the port number of a secondary stream to be used for **stderr**. A second connection is then created to the specified port on the client's machine. The source port of this second connection is also in the 0-1023 range.

4. The server passes the client's source address to *gethostbyaddr*(3B) to obtain the corresponding host name. If the host name cannot be determined, the server will use the dot-notation representation of the Internet address as the client's host name in step 8.

5. A null-terminated user name of 16 characters (maximum) is retrieved on the initial socket. This user name is interpreted as the user identity on the **client**'s machine.

6. A null-terminated user name of 16 characters (maximum) is retrieved on the initial socket. This user name is interpreted as a user identity to use on the **server**'s machine.

7. A null-terminated command to be passed to a shell is retrieved on the initial socket. The command length is limited by the upper boundary on the size of the system's argument list.

8. *rshd* then validates the client-end user from step 5 according to the following steps. The server-end user name from step 6 is looked up in the password file and a *chdir*(2) is performed on the user's home directory. If either the lookup or *chdir*(2) fails, the connection is terminated. If the server-end user does not have a password, the authentication is successful. If the server-end user is not the super-user (user ID 0), the **/etc/hosts.equiv** file is consulted for a list of equivalent hosts. If the client host name (or its alias) is present in this file and the client-end and server-end user names are identical,

the authentication is successful. If the lookup fails or the server-end user is the super-user, the **.rhosts** file in the home directory of the server-end user is checked for the machine name and identity of the client-end user. If this lookup fails, the connection is terminated.

9. A null byte is returned on the initial socket and the command line is passed to the normal login shell of the server-end user. The shell inherits the network connections established by *rshd*.

SEE ALSO

inetd(1M).
rcmd(1), rcp(1), gethostbyaddr(3B), rcmd(3B), services(4) in the *CLIX Programmer's & User's Reference Manual.*

DIAGNOSTICS

Except for the last message listed below, all diagnostic messages are returned on the initial socket. After the messages are returned, any network connections are closed. An error is indicated by a leading byte with a value of 1. (0 is returned in step 9 above on successful completion of all the steps before the execution of the login shell.)

Locuser too long.
: The name of the user on the client's machine is longer than 16 characters.

Remuser too long.
: The name of the user on the remote machine is longer than 16 characters.

Command too long.
: The command line passed exceeds the size of the argument list (as configured into the system).

Login incorrect.
: No password file entry for the user name existed.

No remote directory.
: The *chdir*(2) command to the home directory failed.

Permission denied.
: The authentication procedure described above failed.

Can't make pipe.
: The pipe needed for **stderr** was not created.

Try again.
: A *fork*(2) by the server failed.

Shellname: ...
: The user's login shell could not be started. This message is returned on the connection associated with **stderr** and is not preceded by a flag byte.

CAVEATS

The authentication procedure used here assumes the integrity of each client machine and the connecting medium. This is insecure, but is useful in an open environment.

WARNINGS

If a local (server-end) user does not have a password, the **/etc/hosts.equiv** and **~/.rhosts** files are not checked. Instead, all users from all hosts will be able to gain access as this user. As a security precaution, a system manager should encourage all users to have a password. Users with no password should be given restricted privileges.

NAME

rtape_s - remote tape server

SYNOPSIS

/usr/ip32/inc/rtape_s

DESCRIPTION

rtape_s is a server for the *rtape*(1) program started by the *xns_listener*(1M). *rtape_s* performs a command specified by the *rtape*(1) program. Upon completion of the command, *rtape_s* exits. If a tape error occurs, a message is sent to the *rtape*(1) program. If a network error occurs, a message is printed on the console.

SEE ALSO

xns_listener(1M).
rtape(1), server.dat(4) in the *CLIX Programmer's & User's Reference Manual.*

WARNINGS

This program can only be started by the *xns_listener*(1M).

NAME

rtc_s - remote tape control server

SYNOPSIS

/usr/ip32/inc/rtc_s

DESCRIPTION

rtc_s is a server for the *rtc*(7S) driver, started by the *xns_listener*(1M). *rtc_s* performs various system calls specified by the *rtc*(7S) driver. All system calls performed by the process using the *rtc*(7S) driver are transferred to *rtc_s* and performed on the local *tc*(7S) driver. All errors returned by *tc*(7S) are transferred back to the *rtc*(7S) driver and returned to the process using the *rtc*(7S) driver. Any network errors are printed on the console.

SEE ALSO

rtc(7S), xns_listener(1M).
rtc(1), rtc_allocate(3N), server.dat(4) in the *CLIX Programmer's & User's Reference Manual.*

WARNINGS

This program can only be started by the *xns_listener*(1M).

NAME

runacct - run daily accounting

SYNOPSIS

/usr/lib/acct/runacct [*mmdd* [*state*]]

DESCRIPTION

runacct is the main daily accounting shell procedure. It is normally initiated with *cron*(1M). *runacct* processes connect-time, fee, disk, and process accounting files. It also prepares summary files for *prdaily*(1M) or billing purposes.

runacct attempts to not damage active accounting files or summary files if errors occur. It records its progress by writing descriptive diagnostic messages into the **active** file. When an error is detected, a message is written to **/dev/console**, mail is sent to the root and adm accounts, and *runacct* terminates. *runacct* uses a series of lock files to protect itself from being reinvoked improperly. The files **lock** and **lock1** prevent simultaneous invocation, and **lastdate** prevents more than one invocation per day.

runacct breaks the file processing into separate, restartable states using the file **statefile** to remember the last state completed. It accomplishes this by writing the state name into **statefile**. *runacct* then checks **statefile** to determine its state and to determine what to process next. States are executed in the following order:

SETUP	Move active accounting files into working files.
WTMPFIX	Verify the integrity of the **/etc/wtmp** file, correcting date changes if necessary.
CONNECT1	Produce connect session records in *ctmp* structure format.
CONNECT2	Convert *ctmp* structure format records to *tacct* structure format.
PROCESS	Convert process accounting records to *tacct* structure format.
MERGE	Merge the connect and process accounting records.
FEES	Convert output of *chargefee*(1M) into *tacct* structure format and merge with connect and process accounting records.
DISK	Merge disk accounting records with connect, process, and fee accounting records.
MERGETACCT	Merge the daily total accounting records in **daytacct** with the summary total accounting records in **/usr/adm/acct/sum/tacct**.
CMS	Produce command summaries.

USEREXIT	Include any installation-dependent accounting programs here.
CLEANUP	Clean up temporary files and exit.

To restart *runacct* after a failure, first check the **active** file for diagnostics. Then repair any corrupted data files such as **pacct** or **wtmp**. The **lock**, **lock1**, and **lastdate** files must be removed before *runacct* can be restarted. The argument *mmdd* is necessary if *runacct* is being restarted. *Mmdd* specifies the month and day for which *runacct* will re-run the accounting. The entry point for processing is based on the contents of **statefile**; to override this, include the desired *state* on the command line to designate where processing should begin.

EXAMPLES

To start *runacct*:

```
nohup runacct 2>/usr/adm/acct/nite/fd2log &
```

To restart *runacct*:

```
nohup runacct 0601 2>>/usr/adm/acct/nite/fd2log &
```

To restart *runacct* at a specific state:

```
nohup runacct 0601 MERGE 2>>/usr/adm/acct/nite/fd2log &
```

FILES

/etc/wtmp	login/logout history
/usr/adm/pacct	current progress accounting file
/usr/adm/acct/nite/active	progress record
/usr/adm/acct/nite/daytacct	total accounting records
/usr/adm/acct/nite/lock	lock file
/usr/adm/acct/nite/lock1	lock file
/usr/adm/acct/nite/lastdate	prevents use more than once a day
/usr/adm/acct/nite/statefile	last state completed
/usr/adm/acct/nite/ptacct/*mmdd*	one days records

SEE ALSO

acct(1M), acctcms(1M), acctcon(1M), acctmerg(1M), acctprc(1M), acctsh(1M), fwtmp(1M).
acctcom(1) in the *CLIX Programmer's & User's Reference Manual.*
cron(1M) in the *UNIX System V System Administrator's Reference Manual.*
acct(2), acct(4), utmp(4) in the *UNIX System V Programmer's Reference Manual.*

BUGS

If *runacct* fails in the SETUP state, run SETUP manually. To restart *runacct*, use the following command:

runacct *mmdd* WTMPFIX

If *runacct* failed in the PROCESS state, remove the last **ptacct/***mmdd* file because it will not be complete.

NAME

runcd - mount a CDROM and invoke the CDROM menu

SYNOPSIS

runcd

DESCRIPTION

runcd mounts a CDROM and invokes the CDROM menu. The CDROM menu offers the following choices: view CDROM software delivery documentation; invoke *newprod*(1M); invoke *makenode*(1M); set up System V online documentation and Intergraph online news; exit.

NOTES

Only the super-user may execute this command.

NAME

rwhod - remote system status server

SYNOPSIS

/usr/ip32/tcpip/rwhod

DESCRIPTION

rwhod is the server that maintains the database used by the *rwho*(1) and *ruptime*(1) programs. Its operation is predicated on the ability to broadcast messages on a network.

rwhod may be started on installation of the TCP/IP product by answering the *rwhod* prompt. (The default is off.) Otherwise, to start *rwhod* **/etc/rc2.d/s89tcpip** must be edited to uncomment the *rwhod* startup lines.

rwhod operates as both a producer and consumer of status information. As a producer of information, it periodically queries the state of the system and constructs status messages that are broadcast on the network. As a consumer of information, it listens for other *rwhod* servers' status messages, validates them, and records them in a collection of files located in the directory **/usr/spool/rwho**.

The server transmits and receives messages at the port indicated in the "rwho" service specification; see (*services*(4)). The messages sent and received have the form:

```
struct outmp {
        char    out_line[8];             /* tty name */
        char    out_name[8];             /* user ID */
        long    out_time;                /* time on */
};
struct whod {
        char    wd_vers;                 /* protocol version # */
        char    wd_type;                 /* packet type */
        char    wd_pad[2];
        int     wd_sendtime;             /* time stamp by sender */
        int     wd_recvtime;             /* time stamp by receiver */
        char    wd_hostname[32];         /* host's name */
        int     wd_loadav[3];            /* load average */
        int     wd_boottime;             /* time system booted */
        struct  whoent {
                struct outmp we_utmp;    /* active tty info */
                int     we_idle;         /* tty idle time */
        } wd_we[1024 / sizeof (struct whoent)];
};
```

All fields are converted to network byte order before transmission. The load averages represent 5, 10, and 15 minute intervals before a server's transmission; they are multiplied by 100 for representation in an integer. The host name included is that returned by the *gethostname*(2B) system call, with any trailing domain name omitted. The array at the end of the message contains information about the users logged in to the sending machine. This information includes the contents of the *utmp*(4) entry for each nonidle terminal line and a value indicating the time in seconds since a character was last received on the terminal line.

Messages received by *rwhod* are discarded unless they originated at an *rwhod* port. In addition, if the host's name, as specified in the message, contains any unprintable ASCII characters, the message is discarded. Valid messages received by *rwhod* are placed in files named **whod.***host-name* in the directory **/usr/spool/rwho**. These files contain only the most recent message in the format described above.

Status messages are generated approximately once every three minutes. *rwhod* performs an *nlist*(3C) on **/unix** periodically to guard against the possibility that this file is not the system image currently operating.

SEE ALSO

rwho(1), ruptime(1), gethostname(2B), services(4) in the *CLIX Programmer's & User's Reference Manual.*
utmp(4) in the *UNIX System V Programmer's Reference Manual.*

BUGS

There should be a way to relay status information between networks. Status information should be sent only on request rather than continuously.

NAME

sendmail - send mail over the Internet

SYNOPSIS

/usr/lib/sendmail [*options*] [*address*...]

DESCRIPTION

sendmail sends a message to one or more *address*es, routing the message over the necessary networks. *sendmail* can forward the message to other hosts as necessary to deliver the message to the correct destination.

sendmail is not intended as a user interface utility. Other utilities provide user-friendly front-ends; *sendmail* is used only to deliver preformatted messages (see *smail*(1M)).

With no options, *sendmail* reads its standard input until an end-of-file or a line consisting only of a single dot is reached and sends a copy of the message to all of the addresses listed. *sendmail* determines how to deliver a message by parsing its addresses. If an address has a domain-style syntax as described in *mailaddr*(7B), *sendmail* will attempt to send the message over a TCP connection to a remote *sendmail* daemon. Otherwise, *sendmail* sends the message to another mailer program.

Local addresses are looked up in an alias file and aliased appropriately. Aliasing can be prevented by preceding the address with a backslash (\). By default, the sender is not included in any alias expansions. For example, if **john** sends to **group** and **group** includes **john** in the expansion, the letter will not be delivered to **john**.

The following options are available:

-ba Change to Advanced Research Projects Agency Network (ARPANET) mode. All input lines must end with a CR-LF, and all messages will be generated with a CR-LF at the end. Also, the **From:** and **Sender:** fields are examined for the name of the sender.

-bd Execute as a daemon. *sendmail* will fork and run in the background listening on Transmission Control Protocol (TCP) socket 25 for incoming Simple Message Transfer Protocol (SMTP) connections.

-bi Initialize the alias database.

-bm Deliver a mail message. If a **-b***x* option is not specified, this mode is used by default. If this option is specified after another **-b***x* option, this option overrides the other option. One or more addresses should be specified with this option.

-bp Display a listing of the queue contents.

-bs Use the SMTP protocol as described in RFC821 on standard input and output. All input lines must end with a CR-LF, and all messages will be generated with a CR-LF at the end.

-bt Execute in address test mode. This mode reads addresses and shows the steps taken as they are being parsed. This test mode is used for debugging configuration files.

-bv Verify addresses only. Do not try to collect or deliver a message. Verify mode is normally used for validating addresses or mailing lists. One or more addresses should be specified with this option.

-C*file* Specify *file* as the configuration file. *sendmail* will not run as root if an alternate configuration file is specified.

-d[*list*] Set the debugging level to *list*. If *list* is not specified, a *list* of 0-99.1 is used by default, which sets debug flags 0-99 to level 1.

-F*fname* Set the full name of the sender to *fname*.

-f*name* Set the name of the mail message sender to *name*. This option can be used only by trusted users or by users specifying their own user names. Normally, *sendmail* is configured to define root, daemon, uucp, and network as trusted users.

-h*count* Set the hop count to *count*. The hop count is incremented every time the mail is sent by *sendmail*. When it reaches a limit, the mail is returned with an error message. If the **-h** option is not specified, the **Received:** lines in the message are counted to determine the hop count.

-I Initialize the alias database.

-l*file* Write logging information to *file*. The default is **/dev/console**. This option is effective only when messages are sent.

-n Do not look up local addresses in the alias file.

-q[*time*] Process saved messages in the queue at the interval specified by *time*. If *time* is omitted, process the queue once. If *time* is specified, *sendmail* will run as a background process. *Time* must be specified as a number followed by a character specifying the unit of time. The following characters are valid: **s** for seconds, **m** for minutes, **h** for hours, **d** for days, and **w** for weeks. For example, **-q1h30m** or **-q90m** would both set the interval to one hour and thirty minutes. This option is often used with the **-bd** option.

-r*name* Specify an alternate and obsolete form of the **-f** option.

-t Read the message for recipients. **To:**, **Cc:**, and **Bcc:** lines will be scanned for recipient addresses. The **Bcc:** line will be deleted before transmission. Any addresses in the argument list will be suppressed. That is, they will not receive copies even if they are listed in the message header.

-v Set verbose mode. Alias expansions will be displayed.

-s Same as **-of**.

-c Same as **-oc**.

-e Same as **-oe**.

-i Same as **-oi**.

-m Same as **-om**.

-T Same as **-oT**.

-v Same as **-ov**.

-? Display usage message.

-o*opt arg* Set option *opt* to the specified *arg*. Options can be set either on the command line using the **-o** option or in the configuration file. These options are described in detail in the *Sendmail Administrator's Guide*. The options are as follows:

A*file* Use an alternate alias file. The default is **/usr/lib/aliases**.

a*min* If set, wait a maximum of *min* minutes for an @:@ entry to exist in the alias database before starting. If the entry does not appear in *min* minutes, rebuild the database (if the **D** option is also set) or issue a warning. If no *min* is specified, wait a maximum of five minutes.

B*char* Set the blank substitution character to *char*. Unquoted spaces in addresses are replaced by this character.

c On mailers that are expensive to connect to, immediate connection should not be initiated. Instead, messages are saved in the queue and delivered the next time the queue is processed.

d*mode* Set the delivery mode to *mode*. Valid *mode*s are **i** for interactive (synchronous) delivery, **b** for background (asynchronous) delivery, and **q** for queue only. If the queue mode is specified, actual delivery occurs the next time the queue is processed.

D Rebuild the alias database if changes have been made to **/usr/lib/aliases.**

e*mode* Set error processing mode to *mode*. Valid *modes* are **m** to mail back the error message, **w** to write back the error message (or mail it back if the sender is not logged in), **p** to print the errors on the terminal (default), **q** to throw away error messages (and only an exit status is returned), and **e** to mail back the error messages and return a zero exit status (for Berknet). If the message text is not mailed back by mode **m** or **w** and if the sender is local to this machine, a copy of the message is appended to the file **dead.letter** in the sender's home directory.

F*mode* Set the access mode for creating temporary files to *mode*. This default is 644.

f Save UNIX-style **From:** lines at the front of messages.

g*gid* Set the default group ID (GID) for mailers to *gid*. Mailers without the **S** option in their mailer definition will run as this GID. The default is GID 1.

H*file* Specify the SMTP help file to be *file*. The default is **/usr/lib/sendmail.hf**.

i Do not interpret dots on a line by themselves as a message terminator.

L*level* Set the log level to *level*. *Level* must be between 0 and 22. The default is 0, which means logging is disabled.

l*file* Write logging information to *file*. The default is **/dev/console**.

M*mvalue* Set macro *m* to *value*. This option is intended only for use from the command line. *m* may be any alphabetic character.

m Send to the sender if the sender is in an alias expansion.

N*netname* Specify the name of the home network (ARPA by default). The argument of an SMTP and HELO command is checked against *hostname.netname*, where *hostname* is requested from the kernel for the current connection. If they do not match, **Received:** lines are augmented by the name that is determined in this manner so that messages can be traced accurately.

n Validate the right-hand side of aliases when running *newaliases*(1).

o Indicate that this message might have old style headers (spaces between addresses). If not set, the message will have new style headers (commas between addresses). If set, an adaptive algorithm is used to determine which style is used.

P*addr* Set the postmaster copy address for returned mail to *addr*.

Q*queuedir* Select the directory in which to queue messages to *queuedir*. The default is **/usr/spool/mqueue**.

r*timeout* Set the timeout on reads to *timeout*. *Timeout* must be specified as a number followed by a character specifying the unit of time. The following characters are valid: **s** for seconds, **m** for minutes, **h** for hours, **d** for days, and **w** for weeks. For example, **-or1h30m** or **-or90m** would both set the timeout to one hour and thirty minutes. If this option is not specified, *timeout* is set to 0, meaning that *sendmail* will wait indefinitely for a mailer.

S*file* Save statistics in *file*. The default file is **/usr/lib/sendmail.st**.

s Always put messages in temporary files in the queue directory, even when it is not necessary. This ensures that messages are not lost during a system crash.

T*time* Set the length of *time* that undelivered messages will remain in the queue. *Time* must be specified as a number followed by a character specifying the unit of time. The following characters are valid: **s** for seconds, **m** for minutes, **h** for hours, **d** for days, and **w** for weeks. For example, **-oT1h30m** or **-oT90m** would both set the timeout to one hour and thirty minutes. If a message cannot be delivered within this time (because of a host being down), failed messages will be returned to the sender. The default is three days.

U*file* Specify the file that contains a list of UUCP hosts. The default is **/usr/lib/sendmail.uf**.

u*uid* Set the default user ID (UID) for mailers to *uid*. Mailers without the **S** option in their mailer definition will run as this UID. The default is UID 1.

v	Set verbose mode.
y*fact*	Add the indicated *fact* to the priority (thus lowering the job priority) for each recipient. This value penalizes messages with large numbers of addresses. The default is 1000.
Y	Deliver each job that is run from the queue in a separate process. This option should be used when memory is limited, because the default consumes large amounts of memory while the queue is being processed.
z*fact*	Multiply the indicated *fact* by the message class (determined by the Precedence: header in the message and the **P** lines in the configuration file) and subtract from the priority. Thus, messages with a higher priority will be favored. The default is 1800.
Z*fact*	Add the indicated *fact* to the priority every time a job is processed. Thus, each time a job is processed, its priority will be decreased by the indicated value. In most environments this value should be positive, since hosts that are down are often down for a long time. The default is 9000.

DIAGNOSTICS

sendmail returns an exit status code describing its action. The codes are defined as follows:

EX_OK	All addresses completed successfully.
EX_USAGE	*sendmail* was invoked with incorrect options or arguments.
EX_NOUSER	User name was not recognized.
EX_NOHOST	Host name was not recognized.
EX_UNAVAILABLE	Necessary resources were not available.
EX_SOFTWARE	Internal software error, including bad arguments, has occurred.
EX_OSERR	Temporary operating system error, such as cannot *fork*(2) a child process, has occurred.
EX_OSFILE	Critical system file is missing.
EX_CANTCREAT	Cannot create (user) output file.
EX_IOERR	I/O error has occurred.
EX_TEMPFAIL	Message could not be sent immediately, but was queued.

EX_PROTOCOL	Remote error has occurred in protocol.

FILES

Except for **/usr/lib/sendmail.cf**, the following path names are specified in **/usr/lib/sendmail.cf**. Thus, these paths are only approximations.

/usr/lib/aliases	raw data for alias names
/usr/lib/aliases.pag	database of alias names
/usr/lib/aliases.dir	database of alias names
/usr/lib/sendmail.cf	configuration file
/usr/lib/sendmail.hf	help file
/usr/lib/sendmail.uf	UUCP host file
/usr/lib/sendmail.st	collected statistics
/usr/spool/mqueue/*	temp files

SEE ALSO

smail(1M).
aliases(4), mailaddr(7B) in the *CLIX Programmer's & User's Reference Manual.*
mail(1), mailx(1) in the *UNIX System V User's Reference Manual.*

NAME

sethostd - DNP virtual terminal server

SYNOPSIS

sethostd

DESCRIPTION

sethostd is the Digital Network Protocol (DNP) server process that enables a user on a remote host to log in to the local host. *sethostd* is invoked by *netserver*(1M) on the local host and responds to requests from the *sethost*(1) command of the remote host. *sethost*(1) uses the Command Terminal Protocol (CTERM) to communicate with the server.

SEE ALSO

netserver(1M).
sethost(1) in the *CLIX Programmer's & User's Reference Manual.*

NAME

showmount - show all remote mounts

SYNOPSIS

/etc/showmount [**-a**] [**-d**] [**-e**] [*host*]

DESCRIPTION

showmount lists all clients that have remotely mounted a file system from *host*. This information is maintained by the *mountd*(1M) server on *host* and is saved across crashes in the file **/etc/rmtab**. The default value for *host* is the node name returned by *uname*(1).

The following options are available:

-d List directories remotely mounted by clients.

-a Print all remote mounts in the format

host-name:*directory*

where *host-name* is the name of the client and *directory* is the root of the mounted file system.

-e Print the list of exported file systems.

SEE ALSO

mountd(1M).
rmtab(4), exports(4) in the *CLIX Programmer's & User's Reference Manual.*

CAVEATS

/etc/rmtab is removed each time a server reboots.

NAME

smail, rmail - UUCP mailer with routing

SYNOPSIS

smail [*options*] *address* ...

rmail [*options*] *address* ...

DESCRIPTION

smail/rmail, the UUCP mail transport mechanism for *sendmail*(1M), routes mail messages according to the style of their addresses. *smail* passes messages directly to *uux*(1C) for UUCP style addresses (bang (!) paths) and to *sendmail*(1M) for local recipients, domain-style addresses, and other defined address styles. *rmail* is invoked by user mail programs (like *mailx*(1)) and when the host receives messages from UUCP.

The following options are available:

-A Display the resolved addresses. Do not collect a message or invoke a mailer.

-d Set verbose mode and do not invoke other mailers.

-v Set verbose mode, but still invoke other mailers.

-h *host-name* Set *host-name*. The default is provided by *uname*(2).

-H *host-domain* Set *host-domain*. The default is *host-name*.UUCP, where *host-name* is specified by the **-h** option or provided by *uname*(2).

-F *address* Specify *address* on the **From:** line in locally generated mail.

-n *namelist* Specify the full name database file name. *smail* supports another type of aliasing intended for full name resolution using a sorted file, *namelist*, of name/address pairs. This allows mail to George.P.Burdell@gatech.edu to be delivered appropriately. These aliases are very simple since they are not composed of long lists of recipients for each alias. They are also numerous, since mail to George.P.Burdell may be addressed to Burdell, G.Burdell, George.Burdell, George.P.Burdell, G.P.Burdell, or P.Burdell. This form of aliasing uses a fast searching algorithm, so it keeps resolution time manageable. If this option is not specified, the full name database file name is **/usr/lib/fullnames**.

-m *number* Set the maximum *number* of jobs passed to *uux*(1C) for immediate delivery by a single invocation of *smail*. The default is 2 jobs.

-u *uuxflags* Specify *uuxflags* as the flags passed to *uux*(1C) for remote mail. The flags specified by this option override the flags that *smail* passes th *uux*(1C) by default.

-L Send all addresses to *sendmail*(1M), including UUCP paths, for processing. The **-L** option causes *rmail* to send even explicit UUCP paths to *sendmail*(1M), presumably to use other transport mechanisms.

Addresses

smail/rmail interprets *user@domain* as a domain address, *host!address* as a UUCP path, and anything else to be a local address.

Because *rmail* images on foreign hosts unpredictably interpret mixed UUCP/domain addresses, *smail/rmail* understands *domain!user* to be a domain address and generates *path!domain!user* when mailing to a cognate *smail/rmail* host. For *smail/rmail* to distinguish *domain!user* from UUCP *host!address*, *domain* should contain at least one period. *smail/rmail* gives precedence to **@** over **!** when parsing mixed addresses. Thus, a!b@c is parsed as (a!b)@c, rather than a!(b@c).

Return Paths

smail/rmail collapses **From_** and **>From_** lines to generate a simple from argument, which it can pass to *sendmail*(1M) or use to create its own **From_** line. *smail* constructs a return path from the host name that follows the string "remote from" on each **>From** line. Each host name is separated by a **!** character. If that address is in *user@domain* format, *smail/rmail* rewrites it as *domain!user*. If *domain* refers to the local host, the address is rewritten as *user*. *smail/rmail* also removes redundant information from the **From_** line. For instance:

...!myhost!myhost.mydomain!...

becomes

...!myhost!...

Leading occurrences of the local host name are removed as well.

smail/rmail generates its own **From_** line unless it is passing mail to *sendmail*(1M). In this case, it passes the return path to *sendmail*(1M) through the **-f***from* argument. For UUCP-bound mail, *smail/rmail* appends to the **From_** line the string "remote from *host-name*," where *host-name* is the UUCP name of the local host. This way, **From_** can indicate a valid UUCP path, leaving the sender's domain address in the **From:** header.

Headers

Certain headers (To:, **From:**, **Date:**, etc.) are required by the RFC822 document. If these headers are absent in locally generated mail, *smail* will insert them. Also, a line of trace information, called a **Received:** line, will be inserted at the top of each message.

FILES

/usr/lib/fullnames	full name database

SEE ALSO

mkfnames(1M), sendmail(1M).
mail(1), uux(1) in the *UNIX System V User's Reference Manual.*
uname(2) in the *UNIX System V Programmer's Reference Manual.*

NAME

statd - NFS network status monitor

SYNOPSIS

/etc/statd

DESCRIPTION

statd interacts with *lockd*(1M) to provide the crash and recovery functions for the locking services of the Network File System (NFS).

FILES

/etc/sm
/etc/sm.bak

SEE ALSO

lockd(1M).
statmon(4) in the *CLIX Programmer's & User's Reference Manual.*

BUGS

The site crash is only detected when the site is recovered.

NAME

swap - swap space control

SYNOPSIS

/etc/swap -a *swapdev swaplow swaplen*
/etc/swap -d *swapdev swaplow*
/etc/swap -l

DESCRIPTION

swap enables the system administrator to add a device to the system swap table, remove a device from the system swap table, or list the devices in the system swap table. This utility is useful if an administrator wishes to distribute swap space across many disk drives or needs to increase swap resources for busy systems.

The argument definitions are as follows:

swapdev The name of the device to be added or removed from the system swap table.

swaplow The block number of the first block of the device to be added or removed from the system swap table.

swaplen The size of the device to be added (in 512-byte blocks).

-a Add a device to the system swap table.

-d Remove a device from the system swap table.

-l Print a formatted display of the devices in the system swap table.

SEE ALSO

swap(2I) in the *CLIX Programmer's & User's Reference Manual.*

DIAGNOSTICS

swap returns error code 1 for incorrect command usage and error code 2 for incorrect add/delete command usage. The utility returns error code 3 if a list, add, or delete command fails.

WARNINGS

Contiguous swap space cannot be added to devices already configured in the system swap table. Each additional swap device is treated separately.

NAME

sysadm - menu interface to do system administration

SYNOPSIS

sysadm [*subcommand*]

DESCRIPTION

sysadm, when invoked without an argument, presents a menu of system administration *subcommands*, from which the user selects. If the optional argument is presented, the named *subcommand* is run or the named submenu is presented.

The *sysadm* command may be given a password. See **admpasswd** in the **Subcommands** section.

Subcommands

The following menus of subcommands are available. (The number after each item indicates the level of the menu or subcommand.)

filemgmt (1)

File management menu.

The subcommands in this menu allow the user to protect files on the hard disk file systems by copying them to diskettes and later restoring them to the hard disk by copying them back. Subcommands are also provided to determine the files to keep on diskette based on age or size.

backup (2)

Backup files from integral hard disk to disk or tape.

Backup saves copies of files from the hard disk file systems. The menus provide several options for backup media, including that of backing up a set of CLIX files to an archive file on a VMS™ system or to a tape drive on another system. There are two kinds of backups:

COMPLETE - copies all files (useful in case of serious file system damage)

INCREMENTAL - copies files changed since the last backup

The normal usage is to do a complete backup of each file system and then periodically do incremental backups. Two cycles are recommended (one set of complete backups and several incrementals to each cycle). Files backed up with backup are restored using **restore**.

bupsched (2)

Backup reminder scheduling menu.

Backup scheduling is used to schedule backup reminder messages and backup reminder checks. Backup reminder messages are sent to the console to remind the administrator to backup particular file systems when the machine is shutdown or a reminder check has been run during the specified time period.

Backup reminder checks specify times that the system will check to see if any backup reminder messages have been scheduled.

schedcheck (3)
: Schedule backup reminder checks.

Backup reminder checks are run at specific times to check to see if any reminders are scheduled. The user specifies the times at which the check is to be run. Checks are run for the reminder messages scheduled by **schedmsg**.

schedmsg (3)
: Schedule backup reminder message.

Backup reminder messages are sent to the console if the machine is shutdown or a reminder check has been scheduled. The user specifies the times when it is appropriate to send a message and the file systems to be included in the message.

diskuse (2)
: Display how much of the hard disk is being used.

Diskuse lets the user know what percentage of the hard disk is currently occupied by files. The list is organized by file system names.

fileage (2)
: List files older than a particular date.

Fileage prints the names of all files older than the date specified by the user. If no date is entered, all files older than 90 days will be listed. If no directory is specified to start in, the **/usr/admin** directory will be used.

filesize (2)
: List the largest files in a particular directory.

Filesize prints the names of the largest files starting at a specific directory. If no directory is specified, the **/usr/admin** directory will be used. If the user does not specify how many large files to list, 10 files will be listed.

restore (2)
: Restore files from **backup** and **store** media to disk.

Restore copies files from disks and tapes made by **backup** and **store** back to disk. Individual files, directories of files, or the entire contents of a disk or tape can be restored. The user can restore from both incremental and complete media. The user can also list the names of files stored on the disk or tape.

store (2)
: Store files and directories of files on disk or tape.

Store copies files from the hard disk to disk or tape and allows the user to optionally verify that they worked and to optionally remove them when done. Typically, these would be files that the user wants to archive or restrict access to. The user can store single files and directories of files. Use the **restore** command to put stored files back on the hard disk and to list the files stored.

machinemgmt (1)
: Machine management menu.

  Machine management functions are tools used to operate the machine (turn it off or reboot).

powerdown (2)
: Prepare for shutting off the machine.

  Powerdown will stop all running programs, close any open files, write information to disk (such as directory information), park the heads of the disk drives, and print the message "System halted" on the console. In this state, the machine may be powered down without risk of damaging the file system.

reboot (2)
: Stop all running programs and then reboot the machine.

  Reboot will stop all running programs, close any open files, write information to disk (such as directory information), and reboot the machine. This can be used to escape some types of system problems, such as when a process cannot be killed.

whoson (2)
: Print list of users currently logged on the system.

  Whoson prints the login ID, terminal device number, and sign-on time of all users who are currently using the computer.

syssetup (1)
: System setup menu.

  System setup routines allow the user to tell the computer what its environment looks like, including the date, time, and time zone, the administration and system capabilities to be under password control, etc. The first-time setup sequence is also here.

admpasswd (2)
: Assign or change administrative passwords.

  Admpasswd lets passwords for administrative commands and logins such as setup and sysadm be set or changed.

datetime (2)
: Set the date, time, time zone, and Daylight Savings Time (DST).

  Datetime tells the computer the date, time, time zone, and whether DST is observed. It is normally run once when the machine is first set up. If DST is observed, the computer will automatically start to observe it in the spring and return to standard time in the fall. The machine must be turned off and turned back on to guarantee that

ALL times will be reported correctly. Most are correct the next time the user logs in.

setup (2)
: Set up the machine the first time.

  Setup allows the user to define the first login, to set the passwords on the user-definable administration logins, and to set the time zone for the machine's location.

syspasswd (2)
: Assign system passwords.

  Syspasswd lets the user set system passwords normally reserved for the very knowledgeable user. For this reason, this procedure may assign those passwords, but may not change or clear them. Once set, they may only be changed by the specific login or the root login.

usermgmt (1)
: User management menu.

  These subcommands allow the list of users that can access the machine to be added to, modified, or deleted. They can also be placed in separate groups so that they can share access to files within the group, but protect themselves from other groups.

addgroup (2)
: Add a group to the system.

  Addgroup adds a new group name or ID to the computer. Group names and IDs are used to identify groups of users who desire common access to a set of files and directories.

adduser (2)
: Add a user to the system.

  Adduser installs a new login ID on the machine. A series of questions is asked about the user and then the new entry is made. More than one user can be entered at a time. Once this procedure is finished, the new login ID is available.

delgroup (2)
: Delete a group from the system.

  Delgroup allows groups to be removed from the computer. The deleted group is no longer identified by name. However, files may still be identified with the group ID number.

deluser (2)
: Delete a user from the system.

  Deluser allows users to be removed from the computer. The deleted user's files are removed from the hard disk and their logins are removed from the **/etc/passwd** file.

lsgroup (2)
: List groups in the system.

  Lsgroup will list all groups entered in the computer. This list is updated automatically by **addgroup** and **delgroup**.

lsuser (2)
: List users in the system.

  Lsuser will list all users entered in the computer. This list is updated automatically by **adduser** and **deluser**.

modadduser (2)
: Modify defaults used by adduser.

  Modadduser allows the user to change some defaults used when adduser creates a new login. Changing the defaults does not effect any existing logins, only logins made from this point on.

modgroup (2)
: Change a group on the system.

  Modgroup allows the user to change the name of a group that the user enters when **addgroup** is run to set up new groups.

moduser (2)
: Menu of commands to modify a user's login.

  This menu contains commands that modify the various aspects of a user's login.

chgloginid (3)
: Change a user's login ID.

  This procedure allows the user to change a user's login ID. Administrative and system logins cannot be changed.

chgpasswd (3)
: Change a user's password.

  This procedure allows removal or change of a user's password. Administrative and system login passwords cannot be changed. To change administrative and system login passwords, see the system **setup** menu.

chgshell (3)
: Change a user's login shell.

  This procedure allows the user to change the command run when a user logs in. The login shell of the administrative and system logins cannot be changed by this procedure.

EXAMPLES

sysadm adduser

FILES

/usr/admin	files that support *sysadm*
/usr/admin/menu	directory menu starts in

NOTES

As presently implemented, the **backup** and **restore** facilities of *sysadm* are convenient but inefficient. Moving large amounts of data (hundreds of megabytes) may take an unacceptable length of time. Furthermore, *sysadm* does not support multiple tape volumes. For large disks, using *scpio*(1) on a local tape drive is recommended. *sysadm* will be enhanced in the future to support *scpio*(1).

NAME

sysconfig - system configuration utility

SYNOPSIS

/usr/src/uts/clipper/sysconfig

DESCRIPTION

sysconfig invokes a menu-driven interface for configuring the system kernel. The *sysconfig* menus categorize all of the tunable parameters and configurable drivers and allow easy editing for the parameters and driver selection.

To view the commands available for use in the *sysconfig* utility, press <HELP> or <CONTROL>-H. This list of commands is described in the "System Reconfiguration Tutorial" chapter of the *CLIX System Guide*.

FILES

./master.d/*
./config

SEE ALSO

mkconfig(1M).
master(4) in the *CLIX Programmer's & User's Reference Manual*.
"System Reconfiguration Tutorial" in the *CLIX System Guide*.

WARNINGS

A nonbootable system may result from an errant configuration change.

CAVEATS

sysconfig must be executed from the parent directory of the **master.d** and **build** directories.

NAME

telnetd - TELNET protocol server

SYNOPSIS

/usr/ip32/tcpip/telnetd

DESCRIPTION

telnetd is a server that supports the Defense Advanced Research Project Agency (DARPA) standard TELNET virtual terminal protocol. *telnetd* is invoked by the Internet server (see *inetd*(1M)), normally for requests to connect to the TELNET port as indicated by the **/etc/services** file (see *services*(4)).

telnetd operates by allocating a pseudo-terminal device (see *pty*(7S)) for a client, and then creating a login process that has the slave side of the pseudo-terminal as **stdin**, **stdout**, and **stderr**. *telnetd* manipulates the master side of the pseudo-terminal, implementing the TELNET protocol and passing characters between the remote client and the login process.

When a TELNET session is started, *telnetd* sends TELNET options to the client side indicating a willingness to perform *remote echo* of characters, to *suppress go ahead*, and to receive *terminal type information* from the remote client. If the remote client is willing, the remote terminal type is propagated in the environment of the created login process. The pseudo-terminal allocated to the client is configured to operate in "cooked" mode.

telnetd is willing to perform: *echo*, *binary*, *suppress go ahead*, and *timing mark*. *telnetd* is willing to have the remote client perform: *binary*, *terminal type*, and *suppress go ahead*.

SEE ALSO

inetd(1M), pty(7S), termio(7S).
telnet(1), services(4) in the *CLIX Programmer's & User's Reference Manual.*

BUGS

Some TELNET commands are only partially implemented.

The TELNET protocol allows for the exchange of the number of lines and columns on the user's terminal, but *telnetd* does not use them.

Because of bugs in the original 4.2 Berkeley Software Distribution (BSD) *telnet*(1), *telnetd* performs some dubious protocol exchanges to determine if the remote client is a 4.2 BSD *telnet*(1).

Binary mode has no common interpretation except between similar operating systems (CLIX in this case).

The terminal type name received from the remote client is converted to lowercase.

The *packet* interface to the pseudo-terminal (see *pty*(7S)) should be used for more intelligent flushing of input and output queues.

telnetd never sends TELNET *go ahead* commands.

NAME

tftpd - DARPA *tftp*(1) server

SYNOPSIS

/usr/ip32/tcpip/tftpd

DESCRIPTION

tftpd is a server that supports the Defense Advanced Research Project Agency (DARPA) Trivial File Transfer Protocol (TFTP). *tftpd* operates at the port indicated by the "tftp" service listed in the file **/etc/services** (see *services*(4)).

Using *tftp*(1) does not require an account or password on the remote system. Due to the lack of authentication information, *tftpd* will allow only publicly readable files to be accessed. Files may be written only if they exist and are publicly writable. Note that this extends the concept of "public" to include all users on all hosts that can be reached through the network. This may not be appropriate on all systems and its implications should be considered before enabling the *tftp*(1) service.

SEE ALSO

tftp(1), services(4) in the *CLIX Programmer's & User's Reference Manual.*

NAME

tunefs - tune an existing Fast File System

SYNOPSIS

/etc/tunefs *option* ... {*special* | *filesys*}

DESCRIPTION

tunefs changes the dynamic parameters of the fast file system represented by the *special* device file or the named *filesys*. These parameters affect the layout policies. The parameters to be changed are indicated by the following options. At least one of the following options must appear on the command line.

-a *maxcontig* Specify the maximum number of contiguous blocks allocated before forcing a rotational delay. (See **-d** below.) The default value is one, since most device drivers require an interrupt per disk transfer. Device drivers that can chain several buffers together in a single transfer should set this parameter to the maximum chain length.

-d *rotdelay* Specify expected time (in milliseconds) to service a transfer completion interrupt and initiate a new transfer on the same disk. It is used to decide how much rotational spacing to place between successive blocks in a file.

-e *maxbpg* Specify the maximum number of blocks any single file can allocate from a cylinder group before it is forced to allocate blocks from another cylinder group. This value is typically set to approximately one quarter of the total blocks in a cylinder group. The intent is to prevent any single file from using all the blocks in a single cylinder group, thus degrading access times for all files subsequently allocated in that cylinder group. This limit causes large files to perform long seeks more frequently than if they were allowed to allocate all blocks in a cylinder group before seeking elsewhere. For file systems with exclusively large files, this parameter should be set higher.

-m *minfree* *Minfree* specifies the percentage of space held back from normal users (the minimum free space threshold). The default value used is 10%. This value can be set to zero. However, throughput can decrease by a factor of three over that obtained at a 10% threshold. Note that if the value is raised above the current usage level, users will be unable to allocate files until enough files have been deleted to get under the higher threshold.

-o *mode* Set optimization mode, specified by *mode*, which may be either space or time. The file system either attempts to minimize the time spent allocating blocks or to minimize the space fragmentation on the disk. If the value of *minfree* (see above) is less than 10%, the file system optimizes for space to avoid running out of full-sized blocks. For values of *minfree* greater than or equal to 10%, fragmentation is unlikely to be a problem, and the file system can be optimized for time.

SEE ALSO

newfs(1M), ffsmkfs(1M).
fs(4) in the *UNIX System V Programmer's Reference Manual.*

CAVEATS

This program should work on mounted and active file systems. Because the super-block is not kept in the buffer cache, the changes will take effect only if the program is run on dismounted file systems. To change the root file system, the system must be rebooted after the file system is tuned.

NAME

uxmailr, uxmails - XNS mail transport program

SYNOPSIS

/usr/ip32/inc/uxmailr [**-v**] [**-x**[*level*]] [*node-name* ...]
/usr/ip32/inc/uxmails

DESCRIPTION

uxmailr is a file transport system that transports *uucp*(1C) work files between Intergraph machines on a Xerox Network Systems (XNS) network.

The following options are supported:

-v Print the version string and exit.

-x [*level*] Run in debug mode. *Level* is a number between 1 and 9. The larger the number, the greater the quantity of debug output.

Node-name is the name of a remote system. Multiple *node-names* may be present on the command line. If *node-name* is not specified, *uxmailr* will search the *uucp*(1C) spool directory for all work files and attempt to transfer the files to each machine listed in the directory.

Because *uucp*(1C) needs to know the name of the remote machine before queuing its job, the *uxmailr* will periodically update the **/usr/lib/uucp/Systems** file with all the machine names found in the Intergraph clearinghouse. Since updating the **Systems** file destroys it, a template file called **/usr/lib/uucp/Systems.skel** is copied to the **Systems** file before it is updated. Any information normally kept in the **Systems** file should be kept in the **Systems.skel** file.

uxmails is the server for *uxmailr*. It is only run on the server machine and can only be run by the *xns_listener*(1M).

FILES

/usr/spool/uucp
/usr/spool/uucp/Systems
/usr/spool/uucp/Systems.skel

SEE ALSO

xns_listener(1M).
clh(4) in the *CLIX Programmer's & User's Reference Manual.*
uucico(1M) in the *UNIX System V System Administrator's Reference Manual.*
mail(1), uucp(1C) in the *UNIX System V User's Reference Manual.*

WARNINGS

All permanent changes should be made to the **Systems.skel** file, not the **Systems** file.

BUGS

uxmailr only implements sending *uucp*(1C) work files between machines. No provision has been made for receiving work files.

NAME

xns_listener - XNS listener

SYNOPSIS

/usr/ip32/inc/xns_listener [-l *logfile*]

DESCRIPTION

xns_listener is the master process by which Xerox Network Systems (XNS) servers are initiated on a system. All requests to connect to a server (by clients) are password checked if the *server.dat*(4) file indicates that passwords are to be checked for the account specified. If the -l *logfile* option is specified, all network requests will be logged to the file indicated.

This process is also responsible for the following:

1) Receiving clearinghouse update requests from the network and applying them to the database.

2) Booting devices on the network (i.e., remote routers).

3) Providing basic information about the system (such as %CPU being used and amount of memory being used) for programs like *netmap*(1M).

SEE ALSO

sni_connect(3N), sni_accept(3N), server.dat(4) in the *CLIX Programmer's & User's Reference Manual.*

NAME

xxt_listener - listener for Intergraph XT remote login requests

SYNOPSIS

/usr/ip32/inc/xxt_listener [[**-x***n*] [**-l**] *logfile*]

DESCRIPTION

xxt_listener is started at boot time and awaits a login request from *visit*(1). When a login request is received, a connection to the remote machine is established. The *xxt_listener* passes the network connection to the *xnsxt*(7S) driver and starts a *getty*(1M) on a **/dev/ttn** device. When *getty*(1M) or any program that it executes exits, the *xxt_listener* asks the *xnsxt*(7S) driver if it should restart the program.

The following options are available:

-x*n logfile* Allow the *xxt_listener* to log information about its activity. All information is sent to *logfile*, which can be specified as a file or a device such as **/dev/console** or **/dev/tty**. The *n* argument is a number from 1-9, which indicates how much the *xxt_listener* reports. The lower the number, the less it reports. An option of **-x1** reports only connection information (one line) and fatal errors. An option of **-x9** turns on all debug messages.

-l *logfile* Log only connection and fatal errors to *logfile*. This option is equivalent to using **-x1**.

FILES

/dev/xt	control device to talk to *xnsxt*(7S) driver
/dev/ttn??	network terminal devices

SEE ALSO

xnsxt(7S).
visit(1) in the *CLIX Programmer's & User's Reference Manual.*
getty(1M) in the *UNIX System V System Administrator's Reference Manual.*

DIAGNOSTICS

If the *xxt_listener* aborts, a diagnostic message preceded by the word **panic:** is written into the *logfile*.

NAME

ypinit - build and install YP database

SYNOPSIS

/etc/yp/ypinit -m
/etc/yp/ypinit -s *master-name*

DESCRIPTION

ypinit sets up a Yellow Pages (YP) database on a YP server. It can be used to set up a master or a slave server. After answering prompts, success or failure is reported to the terminal.

ypinit sets up a master server using the simple model in which that server is master to all maps in the database. This is the way to bootstrap the YP system; later the association of maps to masters can be changed. All databases are built from scratch, either from information available to the program at run time, or from the ASCII database files in **/etc**. Further files may be handled by the YP as required by the local environment. All such files should be in their "traditional" form, rather than the abbreviated form used on client machines.

A YP database on a slave server is set up by copying an existing database from a running server. The *master-name* argument should be the host name of YP server (either the master server for all the maps or a server on which the database is up to date and stable).

Refer to *ypfiles*(4) and *ypserv*(1M) for an overview of the YP.

The following options are available:

-m Indicates that the local host is to be the YP master.

-s *master-name*
Sets up a slave database.

FILES

/etc/passwd
/etc/group

SEE ALSO

makedbm(1M), yppush(1M), ypxfr(1M), ypmake(1M), ypserv(1M).
ypfiles(4) in the *CLIX Programmer's & User's Reference Manual.*

NOTES

Only the super-user may execute this command.

NAME

ypmake - rebuild YP database

SYNOPSIS

cd /etc/yp; make [*map*]

DESCRIPTION

The file **Makefile** in **/etc/yp** is used by *make* to build the Yellow Pages (YP) database. With no arguments, *make*(1) creates *dbm* databases for any YP out-of-date maps and then executes *yppush*(1M) to notify slave databases of a change.

If a *map* is supplied on the command line, *make*(1) updates that map only. Typing **make passwd** creates and *yppush*(1M)'s the password database (assuming it is out of date). Likewise, **make hosts** and **make rpc** create and *yppush*(1M) the host and Remote Procedure Call (RPC) files **/etc/hosts** and **/etc/rpc**.

Three special variables are used by *make*(1): DIR, which gives the directory of the source file; NOPUSH, if nonnull, inhibits a *yppush*(1M) of the new database files; and DOM, used to construct a domain other than the master's default domain. The default for DIR is **/etc**, and the default for NOPUSH is the null string.

Refer to *ypfiles*(4) and *ypserv*(1M) for an overview of YP.

SEE ALSO

makedbm(1M), ypserv(1M), yppush(1M).
make(1) in the *UNIX System V Programmer's Reference Manual.*

NAME

yppasswdd - server for modifying YP password file

SYNOPSIS

/etc/yp/yppasswdd *file* [**-m** *arg* ...]

DESCRIPTION

yppasswdd is a server that handles password change requests from *yppasswd*(1). *yppasswdd* changes a password entry in *file*, which is assumed to be in the format of *passwd*(4). An entry in *file* is changed only if the password presented by *yppasswd*(1) matches the encrypted password of that entry.

The only available option is the following:

-m *arg* Perform a *make*(1) in **/etc/yp** after *file* is modified. All *args* are passed to *make*(1).

yppasswdd is not run by default. To enable remote password updating for the Yellow Pages (YP), put an entry for *yppasswdd* in **/etc/init.d/portmap** of the host serving as the master for the YP password file.

EXAMPLES

If the YP password file is stored as **/etc/yp/src/passwd**, to propagate password changes immediately, the server should be invoked as follows:

```
/etc/yp/yppasswdd /etc/yp/src/passwd -m passwd DIR=/etc/yp/src
```

FILES

/etc/yp/Makefile makefile for YP databases

SEE ALSO

ypmake(1M).
yppasswd(1), passwd(4), ypfiles(4) in the *CLIX Programmer's & User's Reference Manual*.

CAVEATS

This server will eventually be replaced with a more general service for modifying any map in YP.

NAME

yppoll - version of a YP map at a YP server host

SYNOPSIS

/etc/yppoll [**-h** *host*] [**-d** *domain*] *mapname*

DESCRIPTION

yppoll asks a *ypserv*(1M) process for the order number and the host that is the master Yellow Pages (YP) server for the named map. If the server is a version 1 YP protocol server, *yppoll* uses the older protocol to communicate with it. In this case, it also uses the older diagnostic messages in case of failure.

The following options are available:

-h *host* — Ask the *ypserv*(1M) process at *host* about the map parameters. If *host* is not specified, the YP server for the local host is used. That is, the default host is the one returned by *ypwhich*(1M).

-d *domain* — Use *domain* instead of the default host domain.

SEE ALSO

ypserv(1M).
ypfiles(4) in the *CLIX Programmer's & User's Reference Manual.*

NAME

yppush - force propagation of a changed YP map

SYNOPSIS

/etc/yppush [**-d** *domain*] [**-v**] *mapname*

DESCRIPTION

yppush copies a new version of a Yellow Pages (YP) map from the master YP server to the slave YP servers. It is normally run only on the master YP server by the **Makefile** in **/etc/yp/** after the master databases are changed. It first constructs a list of YP server hosts by reading the YP map **ypservers** within the *domain*. Keys within the map **ypservers** are the ASCII names of the machines on which the YP servers run.

A "transfer map" request with the information needed by the transfer agent (the program that actually moves the map) is sent to the YP server at each host to call back the *yppush*. When the attempt is completed (successfully or not) and the transfer agent has sent *yppush* a status message, the results may be printed to **stdout**. Messages are also printed when a transfer is not possible (for instance, when the request message is undeliverable or the timeout period on responses has expired).

Note: Only abbreviated mapnames less than 10 characters may be used.

Refer to *ypfiles*(4) and *ypserv*(1M) for an overview of YP.

The following options are available:

-d *domain* Specify a *domain*.

-v Verbose. Cause messages to be printed when each server is called and for each response. If this flag is omitted, only error messages are printed.

FILES

/etc/yp/*domainname*/ypservers.dir
/etc/yp/*domainname*/ypservers.pag

SEE ALSO

ypserv(1M), ypxfr(1M).
ypfiles(4), ypmapxlate(4) in the *CLIX Programmer's & User's Reference Manual*.
"YP Tutorial" in the *CLIX System Guide*.

BUGS

In the current implementation (version 2 YP protocol), the transfer agent is *ypxfr*(1M), which is started by the *ypserv*(1M) program. If *yppush* detects that it is speaking to a version 1 YP protocol server, it uses the older protocol, sending a version 1 YPPROC_GET request and issues a message to that effect. Unfortunately, there is no way of knowing if or when the map transfer is performed for version 1 servers. *yppush* prints a message saying that an "old-style" message has been sent. The system administrator should later check to see that the transfer has actually taken place.

NAME

ypserv - YP server and binder processes

SYNOPSIS

/etc/yp/ypserv

/etc/yp/ypbind

DESCRIPTION

ypserv and *ypbind* are daemon processes that provide a simple network lookup service consisting of databases and processes for the Yellow Pages (YP). The databases are *ndbm*(3B) files in a directory tree rooted at **/etc/yp**. These files are described in *ypfiles*(4). The processes are **/etc/yp/ypserv**, the YP database lookup server, and **/etc/yp/ypbind**, the YP binder. The programmer interface to YP is described in *ypclnt*(3R). Administrative tools are described in *yppush*(1M), *ypxfr*(1M), *yppoll*(1M), *ypwhich*(1M), and *ypset*(1M). Tools to see the contents of YP maps are described in *ypcat*(1M), and *ypmatch*(1). Database generation and maintenance tools are described in *ypinit*(1M), *ypmake*(1M), and *makedbm*(1M).

Both *ypserv* and *ypbind* are daemon processes typically activated at system startup time from **/etc/init.d/portmap**. *ypserv* runs only on YP server machines with a complete YP database. *ypbind* runs on all machines using YP services, both YP servers and clients.

The *ypserv* daemon's primary function is to look up information in its local database of YP maps. The operations performed by *ypserv* are defined for the implementer by the YP protocol specification and for the programmer by the header file **<rpcsvc/yp_prot.h>**. Communication to and from *ypserv* is through Remote Procedure Calls (RPCs). Lookup functions are described in *ypclnt*(3R) and are supplied as C-callable functions in **/usr/lib/libyp.a**. Four lookup functions are performed on a specified map within some YP domain: *Match*, *Get_first*, *Get_next*, and *Get_all*. The *Match* operation accepts a key and returns the associated value. The *Get_first* operation returns the first key-value pair from the map and *Get_next* can be used to enumerate the remainder. *Get_all* ships the entire map to the requester as the response to a single RPC request.

Two other functions supply information about the map rather than about map entries: *Get_order_number* and *Get_master_name*. In fact, both order number and master name exist in the map as key-value pairs, but the server will not return either of them through the normal lookup functions. (If the map is examined with *makedbm*(1M), however, order number and master name will be visible.) Other functions are used within the YP subsystem, and are not of general interest to YP clients. These functions include *Do_you_serve_this_domain?* and *Transfer_map*.

ypbind remembers information that lets client processes on a single node communicate with *ypserv* processes. *ypbind* must run on every machine that has YP client processes; *ypserv* may not be running on the same node, but must be running somewhere on the network.

The information *ypbind* remembers is called a *binding* — the association of a domain name with the Internet address of the YP server and the port on that host at which the *ypserv* process is listening for service requests. Binding is driven by client requests. As a request for an unbound domain comes in, the *ypbind* process broadcasts on the network trying to find a *ypserv* process that serves maps within that domain. Since the binding is established by broadcasting, at least one *ypserv* process must be on every network. Once a domain is bound by a *ypbind*, that same binding is given to every client process on the node. The *ypbind* process on the local node or a remote node may be queried for the binding of a domain by using the *ypwhich*(1) command.

Bindings are verified before they are given out to a client process. If *ypbind* is unable to communicate with the *ypserv* process it is bound to, it marks the domain as unbound, tells the client process that the domain is unbound, and tries to bind the domain once again. Requests received for an unbound domain will fail immediately. In general, a bound domain is marked unbound when the node running *ypserv* crashes or is overloaded. In this case, *ypbind* will to bind any YP server (typically one that is less heavily loaded) available on the network.

ypbind also accepts requests to set its binding for a particular domain. The request is usually generated by the YP subsystem. *ypset*(1M) is a command to access the *Set_domain* facility. It is for fixing problems, not for normal use.

FILES

/etc/yp/ypserv.log	if this file exists, log information is written to it when error conditions arise

SEE ALSO

yppush(1M), ypwhich(1M), ypxfr(1M), ypset(1M).
ypcat(1), ypmatch(1), ypclnt(3R), ypfiles(4) in the *CLIX Programmer's & User's Reference Manual.*
"YP Tutorial" in the *CLIX System Guide.*

NAME

ypset - point *ypbind*(1M) at a particular YP server

SYNOPSIS

/etc/ypset [**-V1** | **-V2**] [**-h** *host*] [**-d** *domain*] *server*

DESCRIPTION

ypset tells *ypbind*(1M) to obtain Yellow Pages (YP) services for the specified *domain* from the *ypserv*(1M) process running on *server*. A *server* that is down or not running *ypserv*(1M) is not discovered until a YP client process tries to get a binding for the domain. At this point, the binding set by *ypset* is tested by *ypbind*(1M). If the binding is invalid, *ypbind*(1M) attempts to rebind for the same domain.

ypset is useful for binding a client node on a broadcast network or on a broadcast network that is not running a YP server host. It also is useful for debugging YP client applications (for instance, where a YP map only exists at a single YP server host).

When several hosts on the local net are supplying YP services, it is possible for *ypbind*(1M) to rebind to another host even while attempting to find out if the *ypset* operation succeeded. That is, **ypset** *host1* can be typed, and then **ypwhich** which replies *host2*. This reply can be confusing. This is a function of the YP subsystem's attempt to load-balance among the available YP servers and occurs when *host1* does not respond to *ypbind*(1M) because it is not running *ypserv*(1M) (or is overloaded), and *host2*, running *ypserv*(1M), gets the binding.

Server indicates the YP server to bind to and can be specified as a name or an Internet Protocol (IP) address. If specified as a name, *ypset* attempts to use YP services to resolve the name to an IP address. This works only if the node has a current valid binding for the domain in question. In most cases, *server* should be specified as an IP address.

Refer to *ypfiles*(4) and *ypserv*(1M) for an overview of YP.

The following options are available:

-V1 Bind *server* for the (old) version 1 YP protocol.

-V2 Bind *server* for the (current) version 2 YP protocol.

If no version is supplied, *ypset*, first attempts to set the domain for the (current) version 2 protocol. If this attempt fails, *ypset* attempts to set the domain for the (old) version 1 protocol.

-h *host* Set *ypbind*(1M)'s binding on *host* instead of locally. *host* can be specified as a name or as an IP address.

-d *domain* Use *domain* instead of the default domain.

SEE ALSO

ypwhich(1M), ypserv(1M).
ypfiles(4) in the *CLIX Programmer's & User's Reference Manual*.

NAME

ypwhich - return the YP server or map master host

SYNOPSIS

/etc/ypwhich [**-d** *domain*] [**-V1** | **-V2**] [*host-name*]
/etc/ypwhich [**-t** *mapname*] [**-d** *domain*] **-m** *mname*
/etc/ypwhich -x

DESCRIPTION

ypwhich tells which Yellow Pages (YP) server supplies YP services to a YP client or which is the master for a map. If invoked without arguments, it gives the YP server for the local machine. If *host-name* is specified, the machine is queried to find out which YP master it is using.

Refer to *ypfiles*(4) and *ypserv*(1M) for an overview of YP.

The following options are available:

-d Use *domain* instead of the default host's domain.

-V1 Which server is serving version 1 YP protocol-speaking client processes?

-V2 Which server is serving version 2 YP protocol client processes?

If neither version is specified, *ypwhich* attempts to locate the server that supplies the (current) version 2 services. If no version 2 server is currently bound, *ypwhich* attempts to locate the server supplying the version 1 services. Since YP servers and YP clients are both backward compatible, knowing the version currently in use is unnecessary.

-t *mapname* Inhibit nickname translation. Useful if there is a mapname identical to a nickname.

-m Find the master YP server for a map. No *host-name* can be specified with **-m**. *Mname* can be a mapname, or a nickname for a map.

-x Display the map nickname table. This lists the nicknames (*mnames*) the command knows of and indicates the *mapname* associated with each nickname.

SEE ALSO

rpcinfo(1M), ypset(1M), ypserv(1M).
ypfiles(4) in the *CLIX Programmer's & User's Reference Manual.*

NAME

ypxfr - transfer a YP map from a YP server

SYNOPSIS

/etc/ypxfr [**-f**] [**-h** *host*] [**-d** *domain*] [**-c**] [**-C** *tid prog ipadd port*] *mapname*

DESCRIPTION

ypxfr moves a Yellow Pages (YP) map to the local host by using normal YP services. It creates a temporary map in the directory **/etc/yp/domain** (which must already exist), fills it by enumerating the map's entries, fetches the map parameters (master and order number) and loads them. It then deletes any old versions of the map and moves the temporary map to the real mapname.

If *ypxfr* is run interactively, it writes its output to the terminal. However, if it is invoked without a controlling terminal and if the log file **/etc/yp/ypxfr.log** exists, it appends all output to that file. Since *ypxfr* is most often run from *cron*(1M) or by *ypserv*(1M), the log file can be used to retain a record of the attempt and the results.

For consistency between servers, *ypxfr* should be run periodically for every map in the YP database. Different maps change at different rates: the **services.byname** map may not change for months at a time, for instance, and may therefore be checked only once daily. It may be known that **mail.aliases** or **hosts.byname** changes several times a day. In such a case, updates might be checked for hourly. A *crontab*(4) entry can be used to perform periodic updates automatically. Rather than having a separate *crontab*(4) entry for each map, commands can be grouped to update several maps in a shell script. Examples (mnemonically named) are in **/etc/yp/ypxfr_1perday.sh**, **/etc/yp/ypxfr_2perday.sh**, and **/etc/yp/ypxfr_1perhour.sh**. They can serve as reasonable first cuts.

Note: Only abbreviated mapnames less than 10 characters may be used.

Refer to *ypfiles*(4) and *ypserv*(1M) for an overview of YP.

The following options are available:

-f Force the transfer to occur even if the version at the master is no more recent than the local version.

-c Do not send a "Clear current map" request to the local *ypserv*(1M) process. Use this flag if *ypserv*(1M) is not running locally at the time *ypxfr* is running. Otherwise, *ypxfr* cannot communicate with the local *ypserv*(1M) and the transfer fails.

-h *host* Get the map from *host*, regardless of what the map says the master is. If *host* is not specified, *ypxfr* asks the YP service for the name of the master, and attempts to obtain the map from there. *Host* may be a name or an Internet address in the form *a.b.c.d*.

-d *domain* — Specify a domain other than the host's default.

-C *tid prog ipadd port* — This option is only for use by *ypserv*(1M). When *ypserv*(1M) invokes *ypxfr*, it specifies that *ypxfr* should call back a *yppush*(1M) process at the host with Internet Protocol (IP) address *ipaddr*, registered as program number *prog*, listening on port *port*, and waiting for a response to transaction *tid*.

FILES

/etc/yp/ypxfr.log
/etc/yp/ypxfr1pdy
/etc/yp/ypxfr2pdy
/etc/yp/ypxfr1phr
/etc/yp/YP_MAP_X_LATE

SEE ALSO

ypserv(1M), yppush(1M).

ypfiles(4), ypmapxlate(4) in the *CLIX Programmer's & User's Reference Manual.*

"YP Tutorial" in the *CLIX System Guide.*

Special Interfaces (7)

NAME

intro - introduction to special files and interfaces

DESCRIPTION

This section describes special files and interfaces. Additions and changes to UNIX System V found in the CLIX System are included. Certain major collections are identified by a letter after the section number:

(7S) These files refer to specific hardware peripherals and CLIX system device drivers. The files in this section include additions and changes to UNIX System V. This section corresponds to section (7) in the *UNIX System V Administrator's Reference Manual.*

(7B) This section describes various Berkeley Software Distribution (BSD) network interfaces available under CLIX. Address formats for the various protocols are discussed where applicable. All interfaces discussed in this section are additions to UNIX System V.

(7A) This section describes the asynchronous interface drivers. These drivers allow a process to have multiple I/O operations in progress simultaneously. All devices discussed in this section are additions to UNIX System V.

SEE ALSO

intro(7S), intro(7B), intro(7A).

7S

NAME

intro - introduction to special files

DESCRIPTION

This section describes special files added to or changed from UNIX System V. For hardware-related files, the names of the entries are generally derived from names for the hardware, not the names of the special files themselves. Characteristics of both the hardware device and the corresponding CLIX system device driver are discussed where applicable.

NAME

adt - audit trail record device

DESCRIPTION

/dev/audit is a special file used to enable and record file system accesses. Auditing is initiated by opening **/dev/audit**. Audit records can then be read by issuing reads to the opened audit device. Only one process may have the audit device open on the system at a time.

The types of events recorded by the audit device are *open*(2), *creat*(2), *unlink*(2), *link*(2), *exec*(2), *mount*(2), *rmount*(2), *umount*(2) and *rumount*(2).

Each audit record consists of a record preamble and a record body. The record types and preamble formats are defined as follows in the include file **<sys/audit.h>**:

```
#define ADT_VERSION     0

#define OPEN            0    /* open(2)/creat(2) */
#define LINK            1    /* link(2) */
#define UNLINK          2    /* unlink(2) */
#define EXEC            3    /* exec(2) and exece(2) */
#define MOUNT           4    /* mount(2) and rmount(2) */
#define UMOUNT          5    /* umount(2) and rumount(2) */

typedef struct preamble {
     long     p_version;          /* version that generated this record */
     short    p_type;             /* type of record to follow */
     cnt_t    p_flen;             /* bytes to next record */
     time_t   p_time;             /* since Jan 1, 1970 */
     char     p_addr[6];          /* hardware id of this machine */
     uid_t    p_uid;              /* users uid */
     gid_t    p_gid;              /* users gid */
     dev_t    p_tty;              /* controlling tty if one */
     char     p_comm[DIRSIZ];     /* command name */
     char     p_error;            /* 0 = success, else errno */
     char     p_pad[3];           /* word boundary */
};
```

The raw audit records have the following formats:

```
struct adt_open {
     uint     mode;
     uint     owner;
     uint     group;
     uint     len;
     char     file[];
};
```

```
struct adt_link {
      uint    len1;
      uint    len2;
      char    files[];
};
struct adt_unlink {
      uint    len;
      char    file[];
};
struct adt_exec {
      char    args[PSARGSZ];     /* psargs not currently supported */
      uint    len;
      char    file[];
};
struct adt_mount {
      uint    flag;              /* local or remote info structure to
      uint    len;                 follow the mntpt */
      char    mntpt[];
};
struct local_mnt {
      uint    len;
      char    special[];
};
struct rfs_mnt {
      uint    dlen;              /* domain.host (RFS) */
      uint    alen;              /* advertise (RFS) */
      char    data[];            /* two null terminated fields
                                    whose offsets are defined above */
};
struct nfs_mnt {
      uint    hlen;              /* length of host name */
      uint    rplen;             /* length of remote path */
      char    data[];            /* two null terminated fields
                                    whose offsets are defined above */
};
struct adt_umount {
      uint    len;
      char    data[];            /* advertise (RFS) or mntpt or special */
};
```

If auditing is being used for system security, it should be initiated as early during system boot as possible and shut down as late as possible during system shutdown.

FILES

/dev/audit audit device

SEE ALSO

auditd(1M).

WARNINGS

Audit records are queued internally to CLIX in a memory heap. If this queue or heap runs out of space, processes generating audit events will block until space becomes available.

NAME

aux - serial interface driver

DESCRIPTION

A system may support three or four serial ports that behave as described in *termio*(7S). The number available depends on the platform and system configuration.

The format of the minor device mask for *aux* is shown below:

Minor Device Format:

7	6	5	4	3	2	1	0
RESERVED				IOP		PORT	

The *ioctl*(2) system call

ioctl(*fildes*, *request*, *arg*)

can be used with the following *requests* (defined in **<sys/aux.h>**):

AUX_IOCTL_MODEM — Read current modem status. Modem status is returned in the integer pointed to by *arg*. Possible bit fields values include:

MODEM_DCD	data carrier detect
MODEM_DSR	data set ready
MODEM_RI	ring indicator

AUX_IOCTL_SET_RTS — Enable/disable RTS/CTS hardware flow control. If *arg* points to a nonzero integer value, RTS/CTS hardware handshake will be enabled.

AUX_IOCTL_GET_RTS — Read RTS/CTS hardware flow control state. A nonzero value will be returned in the integer pointed to by *arg* if RTS/CTS is enabled.

FILES

/dev/tty??

SEE ALSO

termio(7S).
ioctl(2) in the *UNIX System V Programmer's Reference Manual*.

NOTES

All ports support transmit data (TXD) and receive data (RXD). In addition, port 0 supports request-to-send (RTS), clear-to-send (CTS), data carrier detect (DCD), data terminal ready (DTR), data set ready (DSR), and ring indicator (RI). Ports 1 and 2 only support RTS, CTS, and DCD. Port 3 (if available) only supports TXD and RXD.

NAME

cs - console driver

DESCRIPTION

cs is the console interface driver. The *cs* driver supports the features described in *termio*(7S).

cs automatically configures the system console at boot time. On Intergraph graphic workstations, the console is a window on the graphics screen. On 200 series servers, the console is connected to serial port 0 and on 300 and 400 series servers, the console is connected to serial port 2.

FILES

/dev/console

SEE ALSO

aux(7S), termio(7S).

NAME

dc - SCSI disk driver

DESCRIPTION

dc is the Small Computer System Interface (SCSI) disk driver. It supports seven SCSI controllers, each with two logical units for a maximum of 14 physical disks.

A disk may be logically divided into sections called partitions. Each partition has a unique partition number and modifier number. A disk partition header precedes each disk partition (see *diskpar*(4)). Using the size information in each partition's header, *dc* builds partition tables for each hard disk.

A typical special file to access a disk with one of its associated partitions is

/dev/dsk/s1u0p7.3

where the SCSI address for the controller is 1, the logical unit number is 0, the partition number is 7, and the modifier number is 3. The format of the major and minor device mask for *dc* is shown below:

Major Device Format:

7	6	5	4	3	2	1	0
0	1	RESERVED		SCSI ID			UNIT

Minor Device Format:

7	6	5	4	3	2	1	0
PARTITION				MODIFIER			

The block special files **/dev/dsk/s?u?p?.?** provide buffered access to the specified disk. The kernel buffers disk data in the system buffer cache and uses *dc* only if data needs to be read or written to the disk.

The character special files **/dev/rdsk/s?u?p?.?** provide direct access to the specified disk. Data is transferred from the disk directly to user memory. Transfer requests to disk character special files must be in block multiples.

The *ioctl*(2) system call

ioctl(*fildes*, *request*, *arg*)

can be used with the following *requests*:

DC_PARSIZE
: Get the size of a device. *Arg* points to an integer location which is updated with the size of the device in blocks.

DC_REPAR
: Force *dc* to read the partitions on a disk. *Arg* is not used.

DC_SETPAR
: Establish a soft disk partition. This command allows a process to modify or create a software partition entry without actually writing the partition headers on the disk. *Arg* points to a *dc_softpar* structure shown below and defined in

the header file **<sys/dc.h>**.

```
struct dc_softpar {
        unsigned char   par;      /* partition number */
        unsigned char   mod;      /* modifier number */
        int             start;    /* starting block */
        int             size;     /* size in blocks */
};
```

DC_GETPAR Get the disk partition header information. *Arg* points to a *dc_softpar* structure.

On failure, *errno* is set to one of the following values.

[EINVAL] *Request* is not one of the listed commands.

[EFAULT] *Arg* points to an invalid location.

FILES

/dev/dsk/s?u?p?.?
/dev/rdsk/s?u?p?.?

SEE ALSO

diskpar(4) in the *CLIX Programmer's & User's Reference Manual.*
ioctl(2) in the *UNIX System V Programmer's Reference Manual.*

NOTES

A device file with a partition number of 0 and a modifier number of 0 corresponds to the entire disk specified by the SCSI address for the controller and the logical unit number.

NAME

et - STREAMS Ethernet Interface

DESCRIPTION

et STREAMS devices **et0** ... **et***n* provide the service described in the *AT&T Logical Link Interface* for the CLIX Ethernet ports from 0-*n*. CLIX computers may have one or more ports for connecting the computer to Ethernet local area networks.

clone(7) opens may be performed on *et* devices to find the first available free minor device.

Before frames can be transmitted and received on an opened *et* device, an Ethernet type called the Service Access Point (SAP) must be bound to the device. Binding indicates to the device that frames transmitted should contain the specified SAP in the type SAP field of the Ethernet header and that only frames containing the specified SAP in the type field of the Ethernet header should be received on the device.

Once bound, frames may be transmitted on the device. Incoming frames with type fields matching the bound SAP will be received on the device if the address field contains one of the following: the local individual address, the Ethernet broadcast address, or an active Ethernet multicast address. The SAP may be unbound and a new SAP bound without closing the device. When a *close*(2) is performed on the device and a SAP is still bound to the device, the SAP will automatically be unbound as part of *close*(2).

Two special SAPs allow access to a range of SAP values with only one bound stream. The two values *ISO_SAP* and *TRLR_SAP* are defined in **<sys/lihdr.h>**.

When *ISO_SAP* is bound to an *et* device, the type field in the Ethernet header is treated as a *length* field, indicating the number of bytes that follow the Ethernet header in the frame. All frames received by the system with *length* values less than or equal to *MAX_ISO_SAP* are passed upstream on *et* streams bound to *ISO_SAP*. Likewise, all frames transmitted on *et* streams bound to *ISO_SAP* should have the *length* field set to the amount of data following the Ethernet header. On *et* streams bound to *ISO_SAP*, unpredictable results will occur if the SAP indicated in a unit datagram (unitdata) request does not match the length of the data to be sent.

All frames received by the system with Ethernet header type values between *TRLR_SAP* and *MAX_TRLR_SAP* are passed upstream on *et* streams bound to *TRLR_SAP*. Streams bound to *TRLR_SAP* should not be used to transmit frames.

The Logical Link Interface (LLI) accesses *et* services. *putmsg*(2) sends request primitives to *et*. *getmsg*(2) receives acknowledgement and indication primitives from *et*. A primitive is a message passed upstream or downstream on an *et* stream.

The format of the control part of each message that composes an LLI primitive is described by the appropriate structure and constant definition in the file **<sys/lihdr.h>**. The first longword in each LLI primitive's control part is the primitive type identifier field *PRIM_type*. A unique constant definition exists for each of the 10 LLI primitive identifiers, and each primitive structure will always contain the type of primitive identified in the *PRIM_type* field. As a convenience, the union *DL_primitives* defined in **<sys/lihdr.h>** is a union of all LLI primitive control structures.

The primitives that are initiated (sent downstream to *et*) by the *et* user are the information request, bind request, unbind request, and unitdata request.

The primitives that are initiated by *et* (sent upstream to the user) are the information acknowledgement, bind acknowledgement, error acknowledgement, OK acknowledgement, unitdata indication, and unitdata error indication. The primitives initiated by *et* will be sent upstream to the user as the result of a request sent to *et* by the user or, in the case of the unitdata indication, when data has arrived from the network on the SAP bound to the stream.

The only primitives that have a data part associated with the message are the unitdata request and unitdata indication primitives. All the other primitives use only the control part of a message to perform their functions. Fields in any of the primitive structures named *GROWTH* or *FILLER* are present for future expansion of the LLI interface and are ignored by *et*.

Information Request Primitive

The information request primitive requests that an LLI device return information about the size of relevant parameters plus the current state of the device. It is passed to *et* as a priority message. The control part of this message is a buffer containing a *DL_info_req* structure. The information request primitive may be issued on an open *et* stream while the stream is in any state.

Information Acknowledgement Primitive

The information acknowledgement returned by *et* in response to an information request will be a priority message containing the following *DL_info_ack* structure in its control part:

```
struct DL_info_ack {
     ulong   PRIM_type;          /* always DL_INFO_ACK            */
     long    SDU_max;            /* maximum service data unit size */
     long    SDU_min;            /* minimum service data unit size */
     long    ADDR_length;        /* address length                 */
     long    SUBNET_type;        /* subnet type                    */
     long    SERV_class;         /* service class                  */
     long    CURRENTstate;       /* link layer state               */
     long    GROWTH;             /* for future enhancement */
};
```

The *PRIM_type* field in a *DL_info_ack* is always *DL_INFO_ACK*. The fields *SDU_max* and *SDU_min* indicate the maximum and minimum allowed sizes (respectively) of the data part of unitdata request and indication primitives. The field *ADDR_length* returns the size of the Ethernet address that *et* will use for local and remote addresses in unitdata request and indication primitives. The *SUBNET_type* field will indicate the subnetwork type provided by *et*. The *SERVICE_class* field will indicate whether *et* supports different levels of service. The *CURRENT_state* field will hold the value of the state of the *et* device associated with the stream when the information request arrived.

Bind Request Primitive

The bind request primitive requests an LLI device to bind a SAP to the stream and return the entire Ethernet address associated with the stream. It is passed to *et* as a nonpriority message. The control part of this message is a buffer containing a *DL_bind_req* structure. The bind request primitive may be issued only on an open *et* stream that is in the unbound state (*DL_UNBND*). The *LLC_sap* field of the *DL_bind_req* contains the SAP, in host order, that *et* should bind to the stream for the user. Many of the SAP constants are defined in **<sys/lihdr.h>**. These constants all have the *_SAP* suffix. An *et* stream will be in the unbound state immediately after it has been opened. Then, once it has been bound, the *et* stream will enter the unbound state again after an unbind primitive succeeds on the stream.

If the bind request succeeds, *et* will return a bind acknowledgement in a priority message containing a *DL_bind_ack* structure. If the bind fails, *et* will return a negative acknowledgement in a priority message containing a *DL_error_ack* structure.

Bind Acknowledgement Primitive

The *DL_bind_ack* has the following structure:

```
struct DL_bind_ack {
    ulong   PRIM_type;        /* always DL_BIND_ACK           */
    long    LLC_sap;          /* LLC service access point     */
    long    ADDR_length;      /* address length               */
    long    ADDR_offset;      /* address offset               */
    long    GROWTH[2];        /* for future enhancement       */
};
```

The *PRIM_type* field in a *DL_bind_ack* is always *DL_BIND_ACK*. The *LLC_sap* field in the *DL_bind_ack* returns the SAP that was actually bound to the stream and may be different than the one specified in the *DL_bind_request*. The *ADDR_offset* field is the byte offset from the beginning of the message's control part at which the Ethernet address associated with the stream is returned. The *ADDR_length* field indicates the length (in bytes) of the returned Ethernet address associated with the stream.

The format of the returned Ethernet address is the *lli_ud_addr* structure defined in **<sys/lli.h>**. The *host* field of the *lli_ud_addr* structure contains the 48-bit Ethernet host address in network order. The *sap* field of the *lli_ud_addr* structure contains the 16-bit Ethernet type field (or SAP) bound to the stream in network order. (Network order is most-significant-byte first. An example will follow in the sample unitdata request primitive.)

Once a successful bind has been performed on an *et* stream, the stream will be in the idle state (*DL_IDLE*) where it can perform unitdata requests and indications. In other words, it can transmit and receive Ethernet frames.

Unbind Request Primitive

The unbind request primitive requests an LLI device to unbind the SAP previously bound to a stream and return the device to the unbound state. It is passed to *et* as a nonpriority message. The control part of this message is a buffer containing a *DL_unbind_req* structure. The unbind request primitive may be issued only on an *et* stream that is in the idle state.

If the unbind request succeeds, *et* will return an OK acknowledgement in a priority message containing a *DL_ok_ack* structure. If the unbind fails, *et* will return a negative acknowledgement in a priority message containing a *DL_error_ack* structure.

OK Acknowledgement Primitive

The *DL_ok_ack* is always passed upstream in a priority message, which has the following structure:

```
struct DL_ok_ack {
        ulong   PRIM_type;        /* always DL_OK_ACK                      */
        long    CORRECT_prim;     /* correct primitive being acknowledged*/
};
```

The *PRIM_type* field in a *DL_ok_ack* is always *DL_OK_ACK*. The *DL_ok_ack* positively acknowledges primitives that require only a positive or negative acknowledgement. The *CORRECT_prim* field in the *DL_ok_ack* contains the value of the *PRIM_type* field of the primitive being positively acknowledged.

Error Acknowledgement Primitive

The *DL_error_ack* is always passed upstream in a priority message, which has the following structure:

```
struct DL_error_ack {
        ulong   PRIM_type;        /* always DL_ERROR_ACK        */
        long    ERROR_prim;       /* primitive in error         */
        long    LLC_error;        /* LLC error code             */
        long    UNIX_error;       /* UNIX error code            */
};
```

The *PRIM_type* field in a *DL_error_ack* is always *DL_ERROR_ACK*. The *DL_error_ack* negatively acknowledges primitives that require only a positive or negative acknowledgement. The *ERROR_prim* field in the *DL_error_ack* contains the value of the *PRIM_type* field of the primitive being negatively acknowledged. The *LLC_error* field returns the LLC error code indicating the nature of the error. The *UNIX_error* field should be ignored unless the *LLC_error* field is set to *DLSYSERR*. In this case the *UNIX_error* field will contain a CLIX error code indicating the nature of the failure.

Unit Datagram Request Primitive

The unitdata request primitive requests *et* to transmit an Ethernet frame. A unitdata request is passed to *et* as a nonpriority message. The control part of the message is a buffer containing a *DL_unitdata_req* structure followed by an *lli_ud_addr* structure. The *lli_ud_addr* structure contains the destination Ethernet address and Ethernet type value to be used in the frame's Ethernet header. The data part of the message is a buffer containing the data to be sent in the Ethernet frame (between the Ethernet header and the frame check sequence).

The *DL_unitdata_req* structure has the following format:

```
struct DL_unitdata_req {
    ulong   PRIM_type;          /* always DL_UNITDATA_REQ       */
    long    RA_length;          /* remote address length        */
    long    RA_offset;          /* remote address offset        */
    long    SERV_class;         /* service class                */
    long    FILLER[2];          /* to make as big as a DL_uderror_ind*/
};
```

The *PRIM_type* field in a *DL_unitdata_req* is always *DL_UNITDATA_REQ*. The *RA_length* field should be set to the size of the *lli_ud_addr* structure. The *RA_offset* field is the byte offset from the beginning of the control part of the message at which the *lli_ud_addr* structure begins. The *RA_offset* should be at least the size of a *DL_unitdata_req* structure. The *SERV_class* field should be set to the constant *DL_NOSERV*.

The *host* field of the *lli_ud_addr* structure contains the six-byte Ethernet address, in network order, that will be used as the destination address for the frame. The *sap* field of the *lli_ud_addr* structure should contain the two-byte representation, in network order, of the SAP that was bound on the stream. If *ISO_SAP* was bound to the stream, the *sap* field should contain, in network order, the frame length.

When *et* receives a unitdata request primitive, it checks the sizes of the control part of the message, the address size specified by *RA_length*, and the size of the message's data part to verify that they are within appropriate limits. *et* then checks to ensure that the SAP specified in the *sap* field of the *lli_ud_addr* matches the SAP bound on the stream. Or, it ensures that the SAP matches the length of the frame to be sent if the *ISO_SAP* was bound on the stream. If these tests succeed, *et* builds an Ethernet header for the frame to be transmitted. The Ethernet header's destination address is the one specified in the *host* field of the *lli_ud_addr* passed in the message. The Ethernet header's source address will be the one *et* returns with the bind acknowledgement when this stream was bound. The Ethernet header's type field will match the *sap* field of the *lli_ud_addr* passed with the message. This header is prepended to the data part of the message and the frame is sent to the destination host.

If an error is encountered in the processing of a unitdata request, a unitdata error primitive is sent upstream. If no errors are encountered in the processing of a unitdata request, it may be assumed that a best effort was made by *et* to deliver the frame to its destination. No positive acknowledgement is sent upstream in response to a successfully processed unitdata request primitive.

An example of sending a unitdata request primitive is shown in the **EXAMPLES** section.

Unit Datagram Error Primitive

The unitdata error primitive reports to the user any errors in processing unitdata requests. *et* passes it upstream as a priority message. The control part of this message is a buffer containing a *DL_uderror_ind* structure followed by an *lli_ud_addr* structure.

The DL_uderror_ind structure has the following format:

```
struct DL_uderror_ind {
     ulong   PRIM_type;       /* always DL_UDERROR_IND     */
     long    RA_length;       /* remote address length     */
     long    RA_offset;       /* remote address offset     */
     long    SERV_class;      /* service class             */
     long    ERROR_type;      /* error type                */
};
```

The *PRIM_type* field in a *DL_uderror_ind* is always *DL_UDERROR_IND*. The *RA_length* field is the length of the remote address specified in the unitdata request. The *RA_offset* field is the byte offset from the beginning of the message's control part at which the *lli_ud_addr* structure begins. The *SERV_class* is the subnetwork service class of the packet in error. The *ERROR_type* field defines the protocol-dependent error code. Errors returned by *et* are defined constants in **<sys/lihdr.h>** with the *_UDERR* suffix.

The *lli_ud_addr* structure contains the *host* and *sap* fields that were specified in the *lli_ud_addr* structure of the unitdata request that had the error.

Unit Datagram Indication Primitive

The unitdata indication primitive delivers a received Ethernet frame to the user. A unitdata indication is passed upstream as a non-priority message. The control part of the message contains a *DL_unitdata_ind* structure, followed by an *lli_addr* structure, followed by an *lli_ud_addr* structure. The address structures contain the source and destination addresses and Ethernet type from the received frame as described below.

The data part of the message is a buffer containing the data received in the Ethernet frame between the Ethernet header and the frame check sequence.

The *DL_unitdata_ind* structure has the following format:

```
struct DL_unitdata_ind {
     ulong   PRIM_type;       /* always DL_UNITDATA_IND    */
     long    RA_length;       /* remote address length     */
     long    RA_offset;       /* remote address offset     */
```

```
        long    LA_length;        /* local address length      */
        long    LA_offset;        /* local address offset      */
        long    SERV_class;       /* service class             */
    };
```

The *PRIM_type* field in a *DL_unitdata_ind* is always *DL_UNITDATA_IND*. The *RA_length* field will be set to the size of the *lli_addr* structure. The *RA_offset* field is the byte offset from the beginning of the message's control part at which the *lli_addr* structure begins. The *LA_length* field will be set to the size of the *lli_ud_addr* structure. The *LA_offset* field is the byte offset from the beginning of the message's control part at which the *lli_ud_addr* structure begins. The *SERV_class* will be set to the subnetwork service class of the received frame.

The *lli_addr* structure contains the six-byte Ethernet source address, in network order, from the Ethernet header in the received frame.

The *host* field of the *lli_ud_addr* structure contains the six-byte Ethernet destination address, in network order, from the Ethernet header in the received frame. The *sap* field of the *lli_ud_addr* structure contains the two-byte representation, in network order, of the Ethernet type in the received frame.

Once an *et* stream has been successfully bound and enters the *DL_IDLE* state, *et* will deliver inbound frames addressed to the local host and bound SAP upstream as they arrive. The delivery of unitdata indications does not depend on a user's request for their delivery. There is no way, short of unbinding the stream, to keep these frames from arriving. If the user does not remove the frames from the queue as fast as they arrive over the network, the stream head between *et* and the user will queue only *STRHIGH* (defined in **<sys/stream.h>**) bytes worth of frames before becoming full. When the stream head's queue becomes full, *et* will begin discarding incoming frames instead of delivering them upstream. This prevents one *et* stream from using up all of the streams resources on the system. It also means that applications that wish to receive all the Ethernet frames received by *et* on the bound SAP must read the frames from the stream as fast as they arrive.

An example of receiving a unitdata indication primitive is shown in the **EXAMPLES** section.

Input/Output Control Calls

In addition to the services provided by the *AT&T Logical Link Interface*, several functions are provided with the STREAMS I_STR *ioctl*(2) call. These functions specify the Ethernet address to be used as the individual address on an Ethernet port and enable and disable active Ethernet multicast addresses for an Ethernet port. The individual address may be changed only when no SAPs are currently bound. (Even the device used for changing the address may not be bound to a SAP.) These *ioctl*(2) calls use the following *lli_addr* address structure defined in **<sys/lli.h>**:

```
        struct lli_addr {
            unchar  host[6];
```

```
};
```

The address should be filled in network order as shown in one of the following examples.

EXAMPLES

The following is an example of sending an Ethernet frame by formatting a unitdata request primitive and using *putmsg*(2). The code assumes that the stream is already bound to SAP 0x800. The frame to be sent contains 100 bytes of data and will be sent to the remote address 08-00-36-01-02-03.

```
#define UD_REQ_SZ       sizeof(struct DL_unitdata_req)
#define UD_ADR_SZ       sizeof(struct lli_ud_addr)

struct strbuf           ctlbuf;
struct strbuf           databuf;
struct DL_unitdata_req  ud_req;
struct lli_ud_addr      ud_adr;
char                    ctrl_buf[UD_REQ_SZ + UD_ADR_SZ];
char                    data_buf[100];

/*
** fill in the DL_unitdata_req and lli_ud_addr structures
*/

ud_req.PRIM_type = DL_UNITDATA_REQ;
ud_req.RA_length = UD_ADR_SZ;
ud_req.RA_offset = UD_REQ_SZ;
ud_req.SERV_class = DL_NOSRV;
ud_adr.host[0] = 0x08;
ud_adr.host[1] = 0x00;
ud_adr.host[2] = 0x36;
ud_adr.host[3] = 0x01;
ud_adr.host[4] = 0x02;
ud_adr.host[5] = 0x03;
ud_adr.sap[0] = 0x08;
ud_adr.sap[1] = 0x00;

/*
** copy the DL_unitdata_req and lli_ud_addr structures
** into the control buffer ctrl_buf and fill in the
** control and data strbuf structures for the message
*/

memcpy(ctrl_buf, &ud_req, UD_REQ_SZ);
memcpy(&ctrl_buf[UD_REQ_SZ], &ud_adr, UD_ADR_SZ);
ctlbuf.len = UD_REQ_SZ + UD_ADR_SZ;
ctlbuf.buf = ctrl_buf;
databuf.len = 100;
databuf.buf = data_buf;

/*
** call putmsg to send the unitdata request primitive
```

```
*/
if (putmsg(fd, &ctlbuf, &databuf, 0) < 0) {
    perror("putmsg failed");
}
```

The following is an example of receiving an Ethernet frame with *getmsg*(2) and parsing the unitdata indication primitive.

```
/*
** the constant MAX_SDU should reflect the SDU_max value
** returned in an information acknowledgement primitive
*/

#define MAX_SDU          2048
#define UD_IND_SZ        sizeof(struct DL_unitdata_ind)
#define UD_ADR_SZ        sizeof(struct lli_ud_addr)
#define ET_ADR_SZ        sizeof(struct lli_addr)

struct strbuf            ctlbuf;
struct strbuf            databuf;
struct DL_unitdata_ind   ud_ind;
struct lli_ud_addr       ud_adr;
struct lli_addr          et_addr
char             ctrl_buf[UD_IND_SZ + UD_ADR_SZ + ET_ADR_SZ];
char                     data_buf[MAX_SDU];
int              flags;

/*
** fill in the control and data strbuf structures and
** call getmsg to get the unitdata indication primitive
*/

ctlbuf.maxlen = UD_IND_SZ + UD_ADR_SZ + ET_ADR_SZ;
ctlbuf.len = 0;
ctlbuf.buf = ctrl_buf;
databuf.maxlen = MAX_SDU;
databuf.len = 0;
databuf.buf = data_buf;
if (getmsg(fd, &ctlbuf, &databuf, &flags) < 0) {
    perror("getmsg failed");
}
if (flags) {
    printf("unitdata indications use nonpriority messages");
}
if (ctlbuf.len < UD_IND_SZ) {
    printf("not big enough to be a unitdata indication");
}
memcpy(&ud_ind, ctrl_buf, UD_IND_SZ);
if (ud_ind.PRIM_type != DL_UNITDATA_IND) {
    printf("this is not a unitdata indication primitive");
}
```

```
if (ud_ind.RA_length != ET_ADR_SZ) {
    printf("this is not a good remote address length");
}
memcpy(&et_adr, &ctrl_buf[ud_ind.RA_offset], ET_ADR_SZ);
if (ud_ind.LA_length != UD_ADR_SZ) {
    printf("this is not a good local address length");
}
memcpy(&ud_adr, &ctrl_buf[ud_ind.LA_offset], UD_ADR_SZ);
/*
** at this point:
**
**  —  databuf.len contains the length of the Ethernet
**     frame data returned in databuf.buf
**  —  et_adr contains the remote (source) address
**     from the Ethernet header of the received frame
**  —  ud_adr contains the local (destination) address and
**     the SAP from the Ethernet header of the received frame.
*/
```

The following is an example of enabling the multicast address 09-00-36-01-02-03.

```
struct lli_addr   ma;
struct strioctl   sioc;

ma.host[0] = 0x09;
ma.host[1] = 0x00;
ma.host[2] = 0x36;
ma.host[3] = 0x01;
ma.host[4] = 0x02;
ma.host[5] = 0x03;

sioc.ic_cmd = LLI_IOC_ADD_MCAST;
sioc.ic_timout = 0;
sioc.ic_len = 6;
sioc.ic_dp = (char *)(&ma.host[0]);
ioctl(et0_fd, I_STR, &sioc);
```

FILES

/dev/et	special device file for *et*
<sys/lihdr.h>	LLI primitive definitions
<sys/lli.h>	*et* specific definitions

SEE ALSO

clone(7), incd(1M).
open(2), close(2), getmsg(2), ioctl(2), putmsg(2), streamio(7), tren(7S) in the *CLIX Programmer's & User's Reference Manual.*

CAVEATS

On CLIX systems, the **/etc/incd** program is usually responsible for creating the network configuration of streams drivers/modules/multiplexors, including *et*. If the individual address for a given Ethernet port needs to be different than the hardware address, **/etc/incd** can be configured to set the address to the desired value. Manual configuration of the individual address or enabling/disabling multicast addresses on a machine affects all network applications system-wide and should therefore be performed with caution. It is necessary only in rare circumstances.

NAME

fl - floppy disk driver

DESCRIPTION

fl is the floppy disk device driver. A floppy may be logically divided into sections called partitions. Each partition has a unique partition number and modifier number. A floppy partition header is found on the first block of the floppy (see *floppypar*(4)). Using the size information in this header, *fl* builds a partition table for the floppy.

A typical special file to access a floppy with its associated partition is

/dev/dsk/flopp7.3

where the partition number is 7 and the modifier number is 3. The format for the minor device for *fl* is show below:

Minor Device Format:

7	6	5	4	3	2	1	0
PARTITION				MODIFIER			

The block special files **/dev/dsk/flopp?.?** provide buffered access to the floppy. The kernel buffers flopy data in the system buffer cache and uses *fl* only if data needs to be read or written to the floppy.

The character special files **/dev/rdsk/flopp?.?** provide direct access to the floppy. Data is transferred from the floppy directly to user memory. Transfer requests to floppy character special files must be in 512-byte block multiples.

The *ioctl*(2) system call

ioctl(*fildes*, *request*, *arg*)

can be used with the following *requests*:

FL_IOCTL_COMMAND — Send floppy drive a command where *arg* is a pointer to a *flopio* structure as defined in **<sys/fl.h>**.

FL_IOCTL_MOTOR — Turn the floppy drive motor on if *arg* is nonzero. Otherwise, turn the floppy drive motor off.

FL_IOCTL_RESET — Reset the floppy drive.

FL_IOCTL_DENSITY — Set up hardware for high density floppy if *arg* is nonzero. Otherwise, set up the hardware for low density.

On failure, *errno* is set to one of the following values.

[ENXIO] *Request* is not one of the listed commands.

[EFAULT] *Arg* points to an invalid location.

FILES

/dev/dsk/floppy
/dev/rdsk/floppy
/dev/dsk/flopp?.?
/dev/rdsk/flopp?.?

SEE ALSO

floppypar(4) in the *CLIX Programmer's & User's Reference Manual.*
ioctl(2) in the *UNIX System V Programmer's Reference Manual.*

NOTES

A device file with a partition number of 0 and a modifier number of 0 always corresponds to the entire floppy as do the **/dev/[r]dsk/floppy** devices.

NAME

gs - generic SCSI driver

DESCRIPTION

gs is the generic Small Computer Systems Interface (SCSI) driver that allows users to send commands to SCSI devices. The special files in **/dev/gs/*** provide access to a controller at any SCSI address.

To obtain a file descriptor associated with a specific SCSI device, *open*(2) the special file associated with the address of the device. For example, if the device to be addressed resides at SCSI address 2 and a logical unit number (LUN) of 3, the associated special file is **/dev/gs/s2u3**. The format of the minor device number for *gs* is shown below:

Minor Device Format:

7	6	5	4	3	2	1	0
0		SCSI			LUN		

The special file **/dev/gs/scsi** can be used to access any SCSI device. Before operations can be performed on this file descriptor, it must be associated with a specific SCSI address and LUN through *ioctl*(2). This file descriptor can be associated repeatedly.

The *ioctl*(2) system call

ioctl(*fildes*, *request*, *arg*)

can be used for the following requests. The structures described below are defined in **<sys/gs.h>**.

GS_CONN Associate global SCSI device with the minor device indicated by the integer pointed to by *arg*.

GS_CMD Send command to controller. *Arg* points to a *gsioc* structure shown below:

```
struct gsioc {
        unsigned char   *dataaddr;      /* data address */
        unsigned char   *senseaddr;     /* sense address */
        unsigned char   *cmdaddr;       /* command address */
        int             datasize;       /* data size */
        int             sensesize;      /* sense size */
        int             cmdsize;        /* command size */
        char            dir;            /* xfer direction */
        char            reterror;       /* return error */
        int             retdatasize;    /* return data size */
        int             retsensesize;   /* return sense size */
};
```

The *cmdaddr* field contains a pointer to the SCSI command buffer to be sent to the target device. The *dataaddr* field contains a pointer to the data buffer associated with the command.

All size fields indicate requested byte counts.

The *dir* field indicates the direction of the data transfer. B_READ indicates that data is being received from the SCSI device. B_WRITE indicates that data is being sent to the SCSI device. These constants are defined in **<sys/buf.h>**.

All size fields for returned data indicate the actual number of bytes transferred.

Upon normal completion of a SCSI command, *reterror* is set to 0. Otherwise, if the target SCSI device indicates a check condition, the *gs* driver will automatically issue a request sense command (0x03) and transfer the resulting sense data to the address indicated in the *senseaddr* field of the *gsioc* structure. The *reterror* field will indicate one of the following errors defined in **<sys/scsi.h>**:

[SCSI_ERR_TIMEOUT]	A SCSI bus timeout occurred during the request.
[SCSI_ERR_BUSY]	The SCSI device indicated a busy status.
[SCSI_ERR_HARDWARE]	The SCSI bus hardware failed.
[SCSI_ERR_RESERVE]	The SCSI device is currently reserved.
[SCSI_ERR_CHKCONDITION]	The SCSI device indicated a check condition.
[SCSI_ERR_BADCOUNT]	The transfer count is greater than the largest transfer size.
[SCSI_ERR_RESET]	SCSI bus reset occurred during the request.
[SCSI_ERR_PARITY]	A SCSI bus parity error occurred during the request.
[SCSI_ERR_INVALID]	An internal driver error occurred.
[SCSI_ERR_SYNCHRONOUS]	A SCSI bus synchronous transfer error occurred during the request.

Upon failure of either *open*(2) or *ioctl*(2), errno is set to one of the following:

[EINVAL]	*Request* is not a valid value for *ioctl*(2).
[EFAULT]	*Arg* points to a nonwritable memory location for *ioctl*(2).
[EBUSY]	The specified SCSI address or LUN is currently busy for *open*(2) or *ioctl*(2).
[ENODEV]	The specified SCSI address or LUN does not exist for *open*(2) or *ioctl*(2).

FILES

/dev/gs/*

SEE ALSO

dc(7S), tc(7S).

NAME

hsio - high speed I/O driver

DESCRIPTION

The *hsio* driver provides an interface to the Raytheon High-Speed Input/Output (HSIO) board. The special files associated with *hsio* are **/dev/hs***xp* where *x* specifies the HSIO board and *p* specifies the IOCT port on the HSIO board.

Each HSIO board in the system is specified by its position on the Shared Resource (SR) Bus. The HSIO board with the lowest SR Bus slot is referenced by a value of 0 for *x*. The next HSIO board on the bus is address by a value of 1 and so forth. Currently only one HSIO board is supported.

There are four IOCT ports (a, b, c, and d) on each HSIO board which are specified by *p*. IOCT port a (IOCTA) is read only. IOCT port b (IOCTB) is write only. IOCT port c (IOCTC) and IOCT port d (IOCTD) can be read or written. Currently only IOCTC and IOCTD are supported.

The *ioctl*(2) system call

ioctl(*fildes*, *request*, *arg*)

can be used with the following *requests* (defined in **<sys/hs.h>**):

HSIO_BITE — If *arg* is nonzero, the board is reset and put into bite mode. Then a simple bite test is performed. A zero is returned if the test was successful. The board is left in bite mode. If *arg* is zero, the board is taken out of bite mode.

HSIO_RESET — Reset both the board and the driver.

In addition to the standard *ioctl*(2) errors, *errno* may also have the following value:

[EIO] — The board did not pass the bite test.

The *read*(2) system call

read(*fildes*, *buf*, *nbyte*)

can be used to read data from the HSIO board. *Buf* must start on a 4-byte boundary and *nbyte* must be a multiple of 4. The first 4-byte word in *buf* will be the control word received by the IOCT port. The control word has the following format:

MSB 0	1	2	3	4	5	6	7	8	...	30	LSB 31
DEV			MOD			COM					

If MOD is 2, the driver will expect data to follow the control word and will place the data in *buf* following the control word. If the amount of data received exceeds the size of *buf*, the remaining data will be discarded. If no data is available, the read will return immediately with a value of zero.

In addition to the standard *read*(2) errors, *errno* may also have one of the following values:

[EIO] A reset occurred during the *read*(2) or this is the first I/O operation since a reset occurred.

[EACCES] The IOCT is a write only port.

The *write*(2) system call

write(*fildes*, *buf*, *nbyte*)

can be used to transmit data to the HSIO board. *Buf* must start on a 4-byte boundary and *nbyte* must be a multiple of 4. The first 4-byte word in *buf* will be sent as a control word. All remaining bytes will be sent as data.

In addition to the standard *write*(2) errors, *errno* may also have one of the following values:

[EBUSY] The IOCT was busy sending a packet out, receiving a packet, or no First In First Out (FIFO) was available on the board.

[EIO] A reset occurred during the *write*(2) or this is the first I/O operation since a reset occurred.

[EACCES] The IOCT is a read only port.

FILES

/dev/hs*

SEE ALSO

read(2), write(2) in the *CLIX Programmer's & User's Reference Manual*.
ioctl(2) in the *UNIX System V Programmer's Reference Manual*.

NAME

icmp - Internet Control Message Protocol

DESCRIPTION

icmp, the Internet Control Message Protocol (ICMP), provides feedback about problems in the communications environment pertaining to processing Internet Protocol (IP) datagrams. It also supplies some basic utilities such as echo functionality and Internet subnet mask maintenance. While ICMP uses IP for support as if it were a higher-level protocol, it is actually an integral part of IP and the *ip*(7S) STREAMS driver.

ICMP error messages pertaining to datagrams sent by *udp*(7S) can be retrieved with the *t_rcvuderr*(3N) call.

The following ICMP messages are supported:

Echo
: Send a message to a specified host with the intent of the destination host returning the same message to the original source.

Echo Reply
: Send a message in response to an echo request. This message should be identical to the original echo request.

Address Mask Request
: Broadcast a message requesting the subnet mask for an interface.

Address Mask Reply
: Send a message with the correct subnet mask in response to an Address Mask Request (if the interface is designated an authoritative agent).

Timestamp
: Send a message to a specific host requesting a Timestamp Reply. The format of the Timestamp in this implementation is Hz since the last time the system was booted (standard Timestamp is milliseconds since midnight UT).

Timestamp Reply
: Send a message with a completed timestamp in response to a Timestamp message.

Destination Unreachable
: Send an error message to a host to indicate that the destination to which it is trying to send is unavailable. The different types of Destination Unreachable messages include the following:

- net unreachable
- host unreachable
- protocol unreachable
- port unreachable
- fragmentation is needed and the "don't fragment" bit in the IP header is set

- source route failed

Redirect
: Send an error message to inform a host of a better route through which a given destination may be reached.

Source Quench
: Send an error message request that a source will slow down its transmission rate because the destination is being overwhelmed.

Time Exceeded
: Send an error message to a source to indicate one of the following two conditions: time to live for a datagram has been exceeded in transit or fragment reassembly time for a datagram has been exceeded.

Parameter Problem
: Send an error message to a source to indicate that an erroneous value has been detected in a protocol header field. The offset field in the Parameter Problem message is set to indicate the header field in which the error was detected.

FILES

<sys/dod/icmp.h> ICMP definitions and message formats

SEE ALSO

inet(7B), ip(7S).

NAME

id - system board identification driver

DESCRIPTION

id is the system board identification driver. It provides access to the information in each board's ID PROM via the *read*(2) system call. The format of the 32 bytes of identification is shown below.

Byte Offset	Description
0-7	board name/number (in ASCII)
8-15	engineering change order (ECO) bits
16-23	software feature bits
24-25	reserved
26-27	family code
28-30	validation footprint
31	checksum (2's complement)

Two categories of special files are available to access identification information. The special files **/dev/iop_id** (minor device 255) and **/dev/unix_id** (minor device 254) are used to access information on 100 and 200 series workstations and servers. The special files **/dev/sr[0-15]** are used to access information on 300 and 400 series workstations and servers. The minor device number is the slot number of the board.

The *read*(2) system call

read(*fildes*, *buf*, *nbyte*)

is used to retrieve identification information. On failure, *errno* is set to one of the following values.

[EINVAL] The file pointer associated with *fildes* points beyond the end of the device.

[EFAULT] *Buf* points to an invalid location.

FILES

/dev/sr[00-15]	special files for 300/400 series workstations
/dev/iop_id	special file for 100/200 series I/O Processor board
/dev/unix_id	special file for 100/200 series CLIPPER board

SEE ALSO

read(2) in the *CLIX Programmer's & User's Reference Manual*.

NAME

ip - Internet Protocol (IP) STREAMS multiplexor

DESCRIPTION

ip is a "cloneable" (see *clone*(7)) STREAMS multiplexing driver that provides the services of the Internet Protocol (IP) to applications, other protocols on the host machine, and gateway functionality to other hosts and networks. IP is the connectionless network layer protocol of the Department of Defense (DoD) Internet Protocol suite.

ip communicates on its upper streams using the AT&T Transport Provider Interface (TPI). Adherence to TPI allows applications to interface with *ip* using the AT&T Transport Layer Interface (TLI). *ip* provides the T_CLTS connectionless protocol service (as specified by TPI/TLI) to its upstream clients. *ip* communicates on its lower streams using the AT&T Logical Link Interface (LLI), expecting that its lower streams are bound to either DOD_SAP (0x800) or TRLR_SAP (0x1001, with *tren*(7S) pushed) before they are linked.

The address contained in bind requests sent to *ip* should consist of one byte that indicates the protocol type to be bound to the *ip* stream. This byte should correspond to the value desired in the protocol field of the IP header on frames transmitted on the device. This value is also needed in incoming frames that will be received on the stream. Some protocol types are listed in **/etc/protocols**.

Datagrams to be transmitted by *ip* are passed to *ip* by unit datagram requests. If no options are specified in the TLI unit datagram request, a default IP header is prepended to the datagram and sent to the destination address passed in the address field of the unit datagram request. The address format for unit datagram requests without options is the *inet_addr* structure described in **<sys/dod/inet.h>**.

```
struct inet_addr {
        unchar  uc[4];
};
```

The address should be filled in using network order. For example the destination address 129.135.200.7 would be filled in using the following code:

```
struct inet_addr  ina;
ina.uc[0] = 129;
ina.uc[1] = 135;
ina.uc[2] = 200;
ina.uc[3] = 7;
```

ip clients can pass the IP header to be sent with the datagram in the options field of unit datagram requests. If the IP header is passed with the unit datagram request, the destination address is assumed to be in the appropriate place in the IP header field of the unit datagram request's options, not in the address field. The destination address passed in the address field is ignored if options are present. The length of the options passed into *ip* must be the size of the *ip_udopt* structure. The following is the format of the *ip_udopt* structure as defined in **<sys/dod/ip.h>**:

```
struct ip_udopt{
        long    flags;
        struct ip_hdr hdr;
};
```

The *flags* field is a bit field indicating which fields of the IP header have been filled and passed to *ip* in the options field of the unit datagram request. Before transmitting the datagram, *ip* fills any unfilled fields with default values. Bitmasks for the *flags* field are in **<sys/dod/ip.h>**. *ip* checks The IP header fields passed to *ip* for validity. If illegal header field values are passed to *ip*, a unit datagram error indication is sent upstream by *ip* and the bitmask of the first invalid field encountered is returned in the *flags* field of the *ip_udopt* structure.

The destination address field must be filled in and the destination address bit must be set in flags when options are passed in because *ip* will try to use that address and not the nonoption destination address. The IP header length field must be filled in and the corresponding bit must be set in flags when options are passed in. The checksum field of the IP header may not be passed into *ip*. If the fragment field of the IP header is passed to *ip*, the identification field must also be filled in.

IP header options follow the default IP header and may be contained in the options array of the *ip_hdr* structure passed to *ip* in the options part of the unit datagram request. If IP header options are to be transmitted with the frame, the appropriate IP header length must be filled in and passed to *ip*. *ip* will not return a unit datagram error indication if there are errors encountered in the IP header options passed in to *ip*. The offending datagram will be thrown away, and an ICMP error message will be sent to the local *ip*. There is no bit in the *flags* field of the *ip_udopt* structure indicating the presence of IP header options.

The following is the format of the *ip_hdr* structure as defined in **<sys/dod/iph.h>**:

```
struct ip_hdr {
    unchar   ihlen : 4;
    unchar   ver : 4;
    tos_u    tos;
    unchar   tlen[2];
    unchar   id[2];
    unchar   frag[2];
    unchar   ttl;
    unchar   proto;
    unchar   xsum[2];
    ina_t    src;
```

```
        ina_t    dst;
        unchar   options[IPH_MAX_OPT_SZ];
    };
```

The *ip* device supports the transmission of datagrams from one to 10240 bytes long and the reception of datagrams from one to 65535 bytes long. *ip* fragments transmit datagrams that are larger than the maximum frame size for the network that they are to be transmitted on into many adequately sized frames. *ip* also reassembles fragmented datagrams to be received locally.

ip will return all frames received locally with a protocol field in the IP header that matches the protocol bound to the stream. The unit datagram indication returned by *ip* will provide options with each returned datagram containing an *ip_udopt* structure. The *flags* field will be set to IP_HDR_UDOPT (all flags set) and the complete IP header of the received datagram contained in the *hdr* field. The length of the returned options will equal the size of the received IP header. Therefore, the options length may be less than the size of the *ip_udopt* structure if the size of the received IP header was less than the maximum size.

ip supports *ioctl*(2) requests to get and set *ip* configuration parameters and to manipulate and retrieve the routing information tables.

ioctl(2) requests to get and set configuration parameters all use the I_STR *ioctl*(2) described in *streamio*(7). The *ic_cmd* field of the *strioctl* structure should be set to the appropriate command value for the operation to be performed. These command values are defined in **<sys/dod/ip.h>**. The *ic_timeout* field of the *strioctl* structure should be set to the desired timeout. The *ic_len* and *ic_dp* fields of the *strioctl* structure should be the size of the *ip_ifreq* structure defined in **<sys/dod/ip.h>** and a pointer to an *ip_ifreq* structure, respectively, for all but the I_IFGETCONF command. In each case, the *ifr_name* field of the *ip_ifreq* pointed to by *ic_dp* should contain the name of the network interface (*ip* lower stream) to work on. The contents of the *ifr_ifru* union field will be determined by the command performed.

The available commands are as follows:

I_IFSETADDR
: Set the address for an interface. The address for the named interface to use will be set to the value sent to *ip* in the *ifr_ifru.ifru_addr* field.

I_IFGETADDR
: Get the address from an interface. The address that the named interface is currently using will be returned from *ip* in the *ifr_ifru.ifru_addr* field.

I_IFSETBRDADDR
: Set the broadcast address for an interface. The broadcast address for the named interface to use will be set to the value sent to *ip* in the *ifr_ifru.ifru_addr* field.

I_IFGETBRDADDR
: Get the broadcast address from an interface. The broadcast address that the named interface is currently using will be returned from *ip* in the *ifr_ifru.ifru_addr* field.

I_IFSETNETMASK
: Set the subnetwork address mask for an interface. The subnetwork address mask for the named interface to use will be set to the value sent to *ip* in the *ifr_ifru.ifru_addr* field.

I_IFGETNETMASK
: Get the subnetwork address mask from an interface. The subnetwork address mask that the named interface is currently using will be returned to *ip* in the *ifr_ifru.ifru_addr* field.

I_IFSETMETRIC
: Set the routing metric for an interface. The routing metric that the named interface will use will be set to the value sent to *ip* in the *ifr_ifru.ifru_metric* field.

I_IFGETMETRIC
: Get the routing metric from an interface. The routing metric that the named interface is currently using will be returned to *ip* in the *ifr_ifru.ifru_metric* field.

I_IFSETFLAGS
: Set flags on an interface. Set the *flags* field on the named interface to the value sent to *ip* in the *ifr_ifru.ifru_flags* field.

I_IFGETFLAGS
: Get flags from an interface. Return the *flags* field of the named interface in the *ifr_ifru.ifru_flags* field.

The flags that may be set by applications through the *ioctl*(2) mechanism are specified by the constant IPL_VALID_FLAGS in **<sys/dod/ip.h>**. The flags are as follows:

```
#define IPL_UP_FL          0x0001  /* Allow/disallow use of interface */
#define IPL_BCAST_FL       0x0002  /* Indicate broadcast address set up */
#define IPL_NOARP_F        0x0080  /* Use/do not use arp on interface */
#define IPL_MASKREP_FL     0x0100  /* Resp. to ICMP addr. mask requests */
#define IPL_VALID_FLAGS    0x0183
```

The I_IFGETCONF command is used to get the Internet address specified on all *ip* interfaces. To do this, *ic_dp* should point to an array of *ip_ifreq* structures and length should reflect the size of the buffer in bytes. *ip* will return the name and address of each interface presently configured and return the length of the resulting buffer in bytes.

ip also supports *ioctl*(2) requests to allow modification of the routing table maintained in the kernel for *ip*. Routing information *ioctl*(2) requests all use the I_STR *ioctl*(2). The *ic_cmd* field of the *strioctl* structure should be set to the appropriate command value for the operation to be performed.

The routing information requests allow adding and deleting entries in the table, getting individual entries from the table, and retrieving the entire table. The structure used to perform the requests is the *ri_entry* structure defined in **ri.h**, as shown in the following.

```
struct ri_entry {
        ina_t       lan;
        ina_t       rtr;
        short       flags;
        short       proto;
        struct ip_lstr*lstr;
};
```

The *lan* field is the Internet address for the entry. The *rtr* field is the Internet address of the router through which datagrams should be sent that are destined for the local area network (LAN) in the *lan* field. The *proto* field is the number used to represent the protocol that was used to determine the route for the LAN. Constant definitions in *ri.h* exist for all presently known routing information protocols. The *lstr* field is used internally by *ip* and its value in routing entries may not be set by applications.

The *flags* field is a bit field indicating the state of the entry. The RI_LOCAL_FL tells *ip* that the LAN is directly connected to this host and, therefore, datagrams may be sent directly to hosts on that LAN without routing. Other flags should be set only by *ip* itself. The following *ioctl*(2) requests are supported:

I_RIADD
: Add an entry to the routing table. You must be super-user to perform this request. The *ic_len* and *ic_dp* fields should be the size of the *ri_entry* structure and a pointer to an *ri_entry* structure, respectively. The *ioctl*(2) will fail if there are not enough streams resources or there was a conflict with another entry in the table.

I_RIDEL
: Delete an entry from the routing table. You must be super-user to perform this request. The *ic_len* and *ic_dp* fields should be the size of the *ri_entry* structure and a pointer to an *ri_entry* structure, respectively. The *ioctl*(2) will fail if an entry did not exist in the table for the specified LAN or if the RI_LOCAL_FL flag was set in the table entry but not set in the request.

I_RIGET
: Get an entry from the routing table. The *ic_len* and *ic_dp* fields should be the size of the *ri_entry* structure and a pointer to an *ri_entry* structure, respectively. The *lan* field tells *ip* which entry to return. The *ioctl*(2) will fail if an entry did not exist in the table for the specified LAN.

I_RITBL
: Get the routing table. The *ic_len* and *ic_dp* fields should be the size of as many *ri_entry* structures as there are entries in the table and a

pointer to a buffer that large, respectively. The *ioctl*(2) will fail if the buffer is not large enough to hold all the entries, and *errno* will be set to the number of entries currently in the table. Otherwise, the buffer will contain an array of *ri_entry* structures, one for each entry in the table.

ip also supports the ARP *ioctl*(2) requests described in *arp*(7B).

FILES

/dev/ip	special device file for IP
<sys/dod/ip.h>	definitions for *ip* device
<sys/dod/iph.h>	definitions for IP protocol
<sys/dod/ipopt.h>	definitions for IP protocol header options
<sys/dod/icmp.h>	definitions for ICMP protocol
<sys/dod/inet.h>	Internet address definitions
<sys/dod/dod_ut.h>	Internet utility and address definitions
<sys/dod/ri.h>	routing table definitions

SEE ALSO

arp(7B), clone(7), icmp(7S), ifconfig(1M), incd(1M), ioctl(2), et(7S), route(1M), routed(1M), streamio(7), tcp(7S), tren(7S), udp(7S).
Section (3N) in the *CLIX Programmer's & User's Reference Manual.*
Section (3N) in the *UNIX System V Programmer's Reference Manual.*
UNIX System V Network Programmer's Guide.

CAVEATS

incd(1M) is used at boot timea to configure the STREAMS drivers and modules that implement network protocols, including *ip*. Manual configuration is not usually necessary.

routed(1M) is usually responsible for maintaining the kernel routing information tables from information gathered by routing information protocols running on the network. Manual manipulation of these tables is necessary only in rare circumstances or in installations where no routing information protcol is in use.

NAME

ldterm - loadable STREAMS module

DESCRIPTION

ldterm is a STREAMS module that can be pushed onto any stream to perform *termio*(7S) style processing. *ldterm* supports all of the standard *ioctl*(2) calls as described in *termio*(7S). Each *ioctl*(2) is passed downstream so that STREAMS drivers may perform the necessary hardware functions. *ldterm* does not set the *c_cflag* field in the *termio* structure. This field should be set by a driver downstream. The STREAMS driver should fill in the *c_cflag* field on any TCGETA request.

All processed data will be sent downstream as M_DATA messages and all incoming data is expected to be in M_DATA messages.

SEE ALSO

termio(7S), xnsxt(7S).

NOTES

Currently only the *xnsxt*(7S) STREAMS driver uses *ldterm*.

NAME

mem, kmem, odt - core memory

DESCRIPTION

The file **/dev/mem** is a special file that is an image of the core memory of the computer. It may be used, for example, to examine and patch the system.

Byte addresses in **/dev/mem** are interpreted as memory addresses. System registers are not accessible through **/dev/mem**. References to nonexistent locations cause errors.

The file **/dev/kmem** is the same as **/dev/mem** except that kernel virtual memory rather than physical memory is accessed. System registers are not accessible through **/dev/kmem**.

The file **/dev/odt** is the same as **/dev/mem** except that no checks are made on the address for **/dev/odt**. This may result in accesses to system registers or system nonexistent memory errors. Examining and patching system registers is likely to lead to unexpected results.

The per-process data for the current process begins at 0xf0000000.

FILES

/dev/mem
/dev/kmem
/dev/odt

WARNINGS

Since no range checking is performed for accesses to **/dev/odt**, it is possible to panic the system with bad addresses.

NAME

mtio - magnetic tape interface

DESCRIPTION

The special files **/dev/rmt/*[n]** refer to magnetic tape drives. If a default autorewind tape device exists, it should be linked with **/dev/rmt/0m**. If a default no-rewind tape device exists, it should be linked with **/dev/rmt/0mn**. No-rewind tape devices typically end in "n".

Each *write*(2) to a tape creates a record with the requested size on the tape. If end-of-tape (EOT) is encountered, no record is written and the returned value is 0. If an attempt is made to *write*(2) beyond EOT, the returned value is -1. If the data reaches the tape, *errno* is set to ENOSPC. Otherwise, *errno* is set to EIO.

Each *read*(2) reads a record from the tape. If the requested size is larger than the record, the returned value is the record size. If the requested size is smaller than the record, the returned value is the requested size and the remaining data in the record will be skipped. If a file mark is encountered, the return value is 0 and the tape is positioned with the head beyond the file mark. Two file marks encountered in sequence indicate EOT for *read*(2) operations.

When a file open for writing is closed, two file marks are written and the tape is positioned with the head between them. This way, if another file is written to tape, a single file mark separates them. Otherwise, the double file mark indicates EOT for *read*(2) operations.

When an autorewind tape device is closed, the tape is rewound to position the head at the beginning-of-tape (BOT). When a no-rewind tape device is closed, the tape is not moved from its current position.

Additional control over the tape device is possible through *ioctl*(2). Structures and "request" definitions are defined in **<sys/mtio.h>**.

The MTIOCTOP "request" uses the *mtop* structure with fields shown below:

```
short    mt_op;       /* operation as defined below */
daddr_t  mt_count;    /* count argument */
```

where *mt_op* is one of the following:

MTWEOF	Write file mark.
MTFSF	Forward skip file.
MTBSF	Backward skip file.
MTFSR	Forward skip record.
MTBSR	Backward skip record.
MTREW	Rewind tape (*mt_count* is ignored).
MTOFFL	Unload and offline tape (*mt_count* is ignored).
MTRETEN	Retension tape (*mt_count* is ignored).
MTERASE	Erase entire tape (*mt_count* is ignored).
MTDENS	Density select (*mt_count* indicates density).

MTFSEOT Forward skip to end of tape (*mt_count* is ignored).

To receive status information about a tape device, use MTIOCGET. The MTIOCGET "request" uses the *mtget* structure with fields shown below:

```
char    type[60];     /* ASCII string for tape drive type */
int     density;      /* tape drive density or QIC format */
int     recsize;      /* record size (0 indicates variable) */
int     resid;        /* residual from previous operation */
int     leftovers;    /* size of controller cached data */
int     flags;        /* for tape drive states */
```

Flags are defined as follows:

MTWP write protect flag

To reliably support multivolume on all tape devices, it is necessary to inquire about and recover any data that may be cached in the tape controller when EOT is encountered. The MTIOCREC "request" uses the *mtrec* structure with fields shown below:

```
char    **rec_addr;    /* recover buffer address */
int     rec_count;     /* byte count to recover */
int     rec_op;        /* operation as defined below */
```

where *rec_op* is one of the following:

MTLEFTOVERS Inquire about the size of cached data (*rec_addr* is ignored).

MTRECOVER Read back cached data.

To inquire the size of cached data, use the MTLEFTOVERS command. *Rec_count* is filled in by the driver. To recover cached data, use MTRECOVER with *rec_addr* pointing to a buffer large enough to store the cached data.

FILES

/dev/rmt/0m
/dev/rmt/0mn

SEE ALSO

mt(1), read(2), write(2) in the *CLIX Programmer's & User's Reference Manual.*
ioctl(2) in the *UNIX System V Programmer's Reference Manual.*

CAVEATS

Not every *ioctl*(2) operation works with every tape drive. Limitations are a function of the tape controller and driver.

NAME

pop - parallel port driver

DESCRIPTION

pop is the general-purpose interface to the parallel port. It can be used to send PRINT mode data to devices requiring either Versatec or Centronics interfaces. *pop* data can be redirected to different devices by using a MUX device. The minor device number corresponds to the MUX port to select. The format of the minor device is shown below:

Minor Device Format:

7	6	5	4	3	2	1	0
INTER-FACE	RESERVED				MUX PORT		

An INTERFACE value of 0 specifies a Centronics interface; a value of 1 specifies a Versatec interface. If the MUX PORT value is 0, then no MUX control will occur. Otherwise, the value specified in the MUX PORT field will be used as the MUX port to be selected (e.g., a value of 1 corresponds to MUX port 1).

The special files **/dev/vop*** provide an interface to devices requiring Versatec data. The special files **/dev/cop*** provide an interface to devices requiring Centronics data.

The *pop* driver can be opened by multiple processes only if each *open*(2) specifies the same MUX port. If O_FNDELAY is set, the error [EBUSY] will be returned if a different MUX port is opened by another process. If O_FNDELAY is not set, then *open*(2) will block until the driver is closed by all other processes.

The *pop* driver supports output only. No *ioctl*(2) commands are available. The timeout supplied for MUX operations is one minute. The timeout for I/O transfers to the device is nine minutes.

The following errors may be returned in *errno* by the *pop* driver on failure of either the *open*(2), *read*(2), or *close*(2) command:

[ENXIO] The *open*(2) failed because the *pop* driver was not present or the minor device specified an illegal MUX port.

[EIO] The *pop* driver was opened for read or the MUX did not respond when selected. For writes, this error indicates a bad status was returned from the device.

[EBUSY] The *pop* driver is currently opened for a different MUX port. Multiple opens are only valid on the same MUX port.

[EFAULT] The *write*(2) buffer address was not a valid memory address or was not 32-bit aligned.

FILES

/dev/vop*
/dev/cop*

NOTE

On Raster Operation Processor (ROP) graphics based machines, sending Centronics data without using a MUX requires the ROP board to be strapped for Centronics output. When sending data to a device connected to a MUX, always use the Versatec device.

NAME

proc - process file system

DESCRIPTION

proc commands provide a number of data gathering and control functions on **/proc** files. A **/proc** file describes a running process. Its contents correspond to the process virtual address space and may be read or written with the *read*(2) and *write*(2) system calls. The *ioctl*(2) commands described below outline additional actions that may be performed on **/proc** files.

The *ioctl*(2) system call

ioctl(*fildes*, *request*, *arg*)

is used to support the *proc* functions. *Fildes* is an open file descriptor that refers to a **/proc** file. *Request* determines the control function to be performed as described below. *Arg* represents additional information needed by the command. The type of *arg* depends on the *request*, but it is generally an integer or a pointer to a data structure.

Since these *proc* commands are implemented through *ioctl*(2), they are subject to the errors described in *ioctl*(2). *Request*-specific errors are described below.

Command Functions

The following *ioctl*(2) commands apply to all **/proc** files:

PIOCGETPR Copy the kernel *proc* structure for the referenced process into the buffer address supplied in *arg*. This structure is in kernel memory and cannot be reached through *read*(2). The referenced buffer should have the size of the *proc* structure as declared in **<sys/proc.h>**. Including this file will require that the header files **<sys/types.h>**, **<sys/immu.h>**, **<sys/param.h>**, and **<sys/region.h>** also be included. On failure, *errno* is set to one of the following values:

[EFAULT] An error occurred when copying the kernel *proc* structure into the buffer pointed to by *arg*.

[ESRCH] The referenced process no longer exists.

PIOCGETUBLK Copy the kernel *user* structure for the referenced process into the buffer address supplied in *arg*. This structure is in kernel memory and cannot be reached through *read*(2). The referenced buffer should have the size NBPP * USIZE as defined in **<sys/immu.h>** and **<sys/param.h>**. Including these files will require that the header file **<sys/types.h>** also be included. On failure, *errno* is set to the following value:

[EFAULT] An error occurred when copying the kernel *user* structure into the buffer pointed to by *arg*.

PIOCSTOP Send the signal SIGSTOP and wait for the process to enter the stopped state. The value returned by the function indicates the reason the process stopped. The possible reasons for stopping are as follows:

REQUESTED	The process was requested to stop through the PIOCSTOP command.
SIGNALLED	The process stopped due to receipt of a signal. For the process to stop on a particular signal, a PIOCSMASK command must be issued.
SYSENTRY	The process was stopped when it entered a system call. For this to occur, a previous PIOCSENTR command must have been issued.
SYSEXIT	The process was stopped when it exited a system call. For this to occur, a previous PIOCSEXIT command must have been issued.

These return values are defined in **<sys/proc.h>**. In addition, *arg* points to an integer that is updated with additional information on what caused the process to stop. The interpretation of this additional stop value depends on the return value as follows:

Return Value	Stop Value Interpretation
REQUESTED	The value will always be 0.
SIGNALLED	The number of the signal causing the process stop.
SYSENTRY	The number of the system call for which the process stopped on entry.
SYSEXIT	The number of the system call for which the process stopped on exit.

On failure, *errno* is set to the following value:

[ESRCH] The referenced process no longer exists.

PIOCWSTOP Wait for a process to stop. See the PIOCSTOP description on the interpretation of *arg* and the return value. On failure, *errno* is set to the following value:

[ESRCH] The referenced process no longer exists.

[EINTR] The receipt of a signal caused premature return from the system call.

PIOCRUN Make a process runnable after a stop. On failure, *errno* is set to the following value:

[ESRCH] The referenced process no longer exists.

PIOCSMASK Specify, through a bit mask, signals whose receipt will cause the process to enter the stopped state. A mask of zeroes turns off signal tracing. The address of the signal mask is supplied in *arg*. The side effect to this call is that a process will remain traced even after the process requesting the trace has closed the /**proc** file. On failure, *errno* is set to one of the following values:

[EFAULT] An error occurred when copying the signal bit mask from the address specified in *arg*.

[ESRCH] The referenced process no longer exists.

PIOCCSIG Clear all pending signals to a process. The referenced process no longer exists.

PIOCEXCLU Mark the text segment of a process as nonshared so that subsequent write requests will succeed. On failure, *errno* is set to one of the following values:

[EIO] An error occurred when attempting to copy the shared, read-only text segment to a nonshared, writable text segment.

[ESRCH] The referenced process no longer exists.

PIOCOPENT Return a read-only file descriptor for the file containing the process text and data segments. This command allows debuggers to access the symbol table without knowing the executable's path name. On failure, *errno* is set to one of the following values:

[EIO] The user block of the referenced process is not in core.

[ESRCH] The referenced process no longer exists.

PIOCSTR Set the trace bit in the process Processor Status Word register. This will cause the process to receive a SIGTRAP signal after executing the next instruction. This is provided to facilitate process single stepping. On failure, *errno* is set to one of the following values:

[EIO] The user block of the referenced process is not in core.

[ESRCH] The referenced process no longer exists.

PIOCRREGS Read the register set of the referenced process into the supplied buffer. The full register set, including 16 general, 8 floating, and 3 special registers, is read. On failure, *errno* is set to one of the following values:

[EFAULT] An error occurred when copying the process registers into the buffer pointed to by *arg*.

[EIO] The user block of the referenced process is not in core.

[ESRCH] The referenced process no longer exists.

PIOCWREGS Read a register set from the supplied buffer and writes it to the address space of the referenced process. The full register set, including 16 general, 8 floating, and 3 special registers is written. On failure, *errno* is set to the one of the following values:

[EFAULT] An error occurred when copying the process's registers from the buffer pointed to by *arg*.

[EIO] The user block of the referenced process is not in core.

[ESRCH] The referenced process no longer exists.

PIOCSENTER Establish a system call which, upon entry, will cause the process to enter the stopped state. The system call number is supplied in an integer pointed to by *arg*. If the supplied system call number is 0, system call entry tracing will be disabled. On failure, *errno* is set to one of the following values:

[EFAULT] An error occurred when attempting to copy the system call number from the address supplied in *arg*.

[EINVAL] The system call number is out of range.

[EIO] The user block of the referenced process is not in core.

[ESRCH] The referenced process no longer exists.

PIOCSEXIT Establish a system call which, upon exit, will cause the process to enter the stopped state. The system call number is supplied in an integer pointed to by *arg*. If the supplied system call number is 0, system call exit tracing will be disabled. On failure, *errno* is set to one of the following values:

[EFAULT] An error occurred when attempting to copy the system call number from the address supplied in *arg*.

[EINVAL] The system call number is out of range.

[EIO] The user block of the referenced process is not in core.

[ESRCH] The referenced process no longer exists.

PIOCFENTR Specify whether a process should enter the stopped stated when a memory fault occurs. If *arg* is 0 the process will not be stopped, otherwise the process will be stopped.

PIOCFADDR The virtual address of the last page that faulted is copied into the integer pointed to by *arg*. On failure, *errno* is set to the following value:

[EFAULT] An error occurred when copying the address into integer pointed to by *arg*.

PIOCGETPINFO

Copy the kernel *pinfo* structure for the referenced process into the buffer address supplied in *arg*. This structure is in kernel memory and cannot be reached through *read*(2). The referenced buffer should have the size of the *pinfo* structure as declared in **<sys/pinfo.h>**. Including this file will require that the header file **<sys/types.h>** also be included. On failure, *errno* is set to the following value:

[EFAULT] An error occurred when copying the *pinfo* structure into the buffer pointed to by *arg*.

SEE ALSO

intro(2), read(2), signal(2), sigset(2), write(2) in the *CLIX Programmer's & User's Reference Manual.*
close(2), ioctl(2), open(2) in the *UNIX System V Programmer's Reference Manual.*
The "PROC Debugging Tutorial" in the *CLIX System Guide.*

DIAGNOSTICS

Unless otherwise specified above, the return value from *ioctl*(2) is 0 upon success and -1 upon failure with *errno* set as indicated.

NAME

pty - pseudo terminal driver

DESCRIPTION

The *pty* driver supports a device-pair called a "pseudo terminal". A pseudo terminal is a pair of character devices, a "master" device, and a "slave" device. The slave device provides an interface identical to the interface described in *termio*(7S). All other devices that provide this interface have a corresponding hardware device. The pseudo terminal slave device has, instead, another process manipulating it through the master half of the pseudo terminal. Anything written on the master device is given to the slave device as input and anything written on the slave device is presented as input on the master device.

The *ioctl*(2) system call

ioctl(*fildes*, *request*, *arg*)

can be used with the following *requests* which apply only to pseudo terminals:

TIOCSTOP
: Stop output to a terminal (as typing <CONTROL>-S). This command does not require a parameter.

TIOCSTART
: Restart output (stopped by TIOCSTOP or by typing <CONTROL>-S). This command does not require a parameter.

TIOCPKT
: Enable/disable packet mode. Packet mode is enabled by specifying (by reference) a nonzero parameter and disabled by specifying (by reference) a zero parameter. When applied to the master side of a pseudo terminal, each subsequent *read*(2) from the terminal will return data written on the slave part of the pseudo terminal preceded by a zero byte (symbolically defined as TIOCPKT_DATA), or a single byte reflecting control status information. In the latter case, the byte is an inclusive-or of zero or more of the bits:

TIOCPKT_FLUSHREAD	The read queue for the terminal is flushed.
TIOCPKT_FLUSHWRITE	The write queue for the terminal is flushed.
TIOCPKT_STOP	Output to the terminal is stopped by <CONTROL>-S.
TIOCPKT_START	Output to the terminal is restarted.
TIOCPKT_DOSTOP	IXON is enabled.
TIOCPKT_NOSTOP	IXON is not enabled.

While this mode is in use, the presence of control status information to be read from the master side may be detected by a *select*(2B) for exceptional conditions.

This mode is used by *rlogin*(1) and *rlogind*(1M) to implement a remote-echoed, locally <CONTROL>-S and <CONTROL>-Q flow-controlled remote login with proper back-flushing of output; it can be used by similar programs.

TIOCUCNTL — Enable/disable a mode that allows a small number of simple user *ioctl*(2) commands to be passed through the pseudo-terminal, using a protocol similar to that of TIOCPKT. The TIOCUCNTL and TIOCPKT modes are mutually exclusive. This mode is enabled from the master side of a pseudo terminal by specifying (by reference) a nonzero parameter and disabled by specifying (by reference) a zero parameter. Each subsequent *read*(2) from the master side will return data written on the slave part of the pseudo terminal preceded by a zero byte or a single byte reflecting a user control operation on the slave side. A user control command consists of a special *ioctl*(2) operation with no data; the command is given as UIOCCMD(*n*), where *n* is a number in the 1-255 range. The operation value *n* will be received as a single byte on the next *read*(2) from the master side. The *ioctl*(2) UIOCCMD(0) is a no-op that may be used to probe for this facility. As with TIOCPKT mode, command operations may be detected with a *select*(2B) for exceptional conditions.

TIOCREMOTE — A mode for the master half of a pseudo terminal, independent of TIOCPKT. This mode causes input to the pseudo terminal to be flow-controlled and not input edited (regardless of the terminal mode). Each write to the control terminal produces a record boundary for the process reading the terminal. In normal use, a write of data is like the data typed as a line on the terminal; a write of 0 bytes is like typing an end-of-file character. TIOCREMOTE can be used when performing remote line editing in a window manager, or whenever flow-controlled input is required.

FILES

/dev/pty[p-r][0-9a-f]	master pseudo terminals
/dev/tty[p-r][0-9a-f]	slave pseudo terminals

SEE ALSO

termio(7S).
read(2), write(2) in the *CLIX Programmer's & User's Reference Manual.*
ioctl(2) in the *UNIX System V Programmer's Reference Manual.*

NAME

rtc - remote tape control STREAMS driver

DESCRIPTION

rtc allows access to a tape drive on another machine as if it resided locally. Functionally, it looks the same as a local tape device.

There are three special files for every *rtc* tape unit: a rewind device, a no-rewind device, and a control device. The rewind device and the no-rewind device behave exactly like a normal tape device. The control device is used to set up or close the connection to the remote machine.

Because allocating a tape drive on another machine may prevent its use by another process, a timeout is set at the end of each access. If the timeout expires, a warning is printed on the console. If the tape device is accessed within the next two minutes, the timeout is restarted. Otherwise, the connection is closed.

FILES

/dev/rmt/rt?	rewind tape device
/dev/rmt/rt?n	no-rewind tape device
/dev/rmt/rt?.ctl	control device

SEE ALSO

rtc_s(1M), tc(7S).
rtc(1), rtc_allocate(3N) in the *CLIX Programmer's & User's Reference Manual*.

WARNINGS

Some older *rtc_s*(1M) servers support only 10K bytes (20 blocks) of data in a single read or write operation.

The *rtc* control device needs to remain open while the tape drive is allocated. If the control device is closed, the connection to the machine with the tape drive will be automatically closed.

NAME

rts - remote terminal server

DESCRIPTION

rts allows remote logins using the Intergraph XT protocol or the Bridge Virtual Terminal protocol. A connection is made from a remote machine by the *visit*(1) program. The listener process, *xxt_listener*(1M), on the local machine handles the establishment of the connection. When the connection is established, the listener transfers the connection to the *rts* driver and starts a *getty*(1M) on a port assigned by the driver. When *getty*(1M) or any program that *getty*(1M) execs terminates, the listener queries the *rts* driver whether the *getty*(1M) should be restarted.

FILES

/dev/ttn??	network terminals
/dev/xt	control device for Intergraph XT protocol
/dev/vtp	control device for Bridge Virtual Terminal protocol

SEE ALSO

xxt_listener(1M), termio(7S).
visit(1) in the *CLIX Programmer's & User's Reference Manual.*
getty(1M) in the *UNIX System V System Administrator's Reference Manual.*

NAME

sb - SDLC/BISYNC communication driver

DESCRIPTION

sb supports communications with a VME board running Synchronous Data Link Control (SDLC) or BISYNC protocol. All data transferred between the board and driver uses the *ioctl*(2) system call. The general format of the *ioctl*(2) command is as follows:

ioctl(*fildes*, *request*, *arg*)

The following is a list of *requests* supported by *sb*:

S_GET
: Receive data from the SDLC/BISYNC communication board. The *ioctl*(2) will return immediately, and if data is present it is returned in the buffer pointed to by *arg*. The first two bytes of the data are an inclusive length. If no data is present, a -1 is returned with *errno* set to EBUSY.

S_GIVE
: Send the buffer pointed to *arg* to the SDLC/BISYNC communication board. The first two bytes of data are an inclusive length.

S_SIGACK
: Enable signal number *arg*. The controlling process will receive the signal specified by *arg* when data is present.

S_RESET
: Reset the SDLC/BISYNC communication board. *Arg* is ignored.

S_CHECKSUM
: Compute and return the checksum for the code loaded on the SDLC/BISYNC communication board. If the checksum is 0, the load was successful. The checksum is returned in an unsigned short pointed to by *arg*.

Addition *requests* use the following structure:

```
typedef struct {
        char    *b_addr;         /* pointer to data buffer */
        int     b_len;           /* byte length of data buffer */
        int     b_load_addr;     /* buffer load address */
} st_boot_rec;
```

The *b_addr* member is a pointer to a data buffer to be sent to the SDLC/BISYNC communication board with length specified by *b_len*. The memory address in the SDLC/BISYNC communication board to begin the load of the buffer is specified by *b_load_addr*.

The *requests* that use this structure are as follows:

S_HEADER
: Send a boot header to the SDLC/BISYNC communication board. This is used to calculate information about the code to be loaded. *Arg* points to a *st_boot_rec* structure. The *b_load_addr* member is not used.

S_BOOT_REC
: Send a data buffer containing code to the SDLC/BISYNC communication board. *Arg* points to a *st_boot_rec*

structure.

FILES

/dev/sb? device files for the SDLC/BISYNC communication board

WARNINGS

All data sent and received from the SDLC/BISYNC communication board must be less than 4K bytes and must be aligned on a 4-byte boundary.

NAME

sc - Optronics ESCAN 200 scanner driver

DESCRIPTION

sc is the Optronics/(rg ESCAN 200 scanner driver. Scanned data is read through the *read*(2) system call. Since scanners are sequential-access devices, *sc* always starts the read of the next scan line during a user's current read. This double buffering allows greater scanner throughput. The maximum size *read*(2) is SC_DATA_SIZE bytes.

The *ioctl*(2) system call

ioctl(*fildes*, *request*, *arg*)

can be used to configure the scanner and to retrieve scanner information. *Arg* points to an *scio* structure with the following members.

```
unsigned char    arg1;       /* first argument of ioctl */
unsigned char    arg2;       /* second argument of ioctl */
unsigned char    arg3;       /* third argument of ioctl */
unsigned char    arg4;       /* fourth argument of ioctl */
unsigned char    buf[18];    /* results buffer */
```

sc updates the array *buf* with the results of the ESCAN operation. This structure is required when using the following *request*s:

REQ_STAT	Get scanner status information. The *arg** members are not used.
REQ_REGS	Get scanner internal register values. The *arg** members are not used.
SET_DATA_GRAY	Force scanner to return gray scale data during its scanning phase. The *arg** members are not used.
SET_DATA_BIN	Force scanner to return binary data during its scanning phase. The *arg** members are not used.
PEJECT	Abort the scan in progress and eject the paper from the scanner. The *arg** members are not used.
RET_VERS	Get the scanner's EPROM revision level. The *arg** members are not used.
SET_200DPI	Set the scanner's resolution to 200 dots per inch. The *arg** members are not used.
SET_400DPI	Set the scanner's resolution to 400 dots per inch. The *arg** members are not used.
SET_THRSH	Set the scanner's threshold value. *Arg1* contains the threshold value. Other *arg** members are not used.
SET_GAIN_1	Set the gain for camera 1 in the scanner. *Arg1* contains the gain value. Other *arg** members are not used.
SET_GAIN_2	Set the gain for camera 2 in the scanner. *Arg1* contains the gain value. Other *arg** members are not used.

SET_GAIN_3 — Set the gain for camera 3 in the scanner. *Arg1* contains the gain value. Other *arg** members are not used.

SET_GAIN_4 — Set the gain for camera 4 in the scanner. *Arg1* contains the gain value. Other *arg** members are not used.

SET_OVLP_1 — Set the overlap for camera 1 in the scanner. *Arg1* contains the overlap value. Other *arg** members are not used.

SET_OVLP_2 — Set the overlap for camera 2 in the scanner. *Arg1* contains the overlap value. Other *arg** members are not used.

SET_OVLP_3 — Set the overlap for camera 3 in the scanner. *Arg1* contains the overlap value. Other *arg** members are not used.

SET_OVLP_4 — Set the overlap for camera 4 in the scanner. *Arg1* contains the overlap value. Other *arg** members are not used.

SET_SCAN_T — Set the scanner's integration time. *Arg1* contains the integration time value. Other *arg** members are not used.

SET_RMRG_BIN — Set the right margin for binary data scanning. *Arg1* contains the least significant byte of the margin value and *arg2* contains the most significant byte of the margin value. Other *arg** members are not used.

SET_RMRG_GRAY — Set the right margin for gray scale data scanning. *Arg1* contains the least significant byte of the margin value, and *arg2* contains the most significant byte of the margin value. Other *arg** members are not used.

Scanner control operations require a different control structure than the *scio* structure described above. When initiating one of these *ioctl*(2) operations, *arg* points to an *sciocomp* structure with the following members.

```
unsigned char   *buf;    /* pointer to compensation buffer */
unsigned int    cnt;     /* size of compensation buffer */
```

The *requests* which require the *sciocomp* structure are listed below:

SET_COMP200 — Set the scanner's stepper motor compensation table for 200 dots per inch scanning.

SET_COMP400 — Set the scanner's stepper motor compensation table for 400 dots per inch scanning.

On failure, *errno* is set to one of the following values.

[EBUSY] The scanner is busy scanning.

[EFAULT] *Arg* points to an invalid location.

[EINVAL]	*Request* is not one of the listed requests.
[EIO]	An error occurred when the command was transferred to the scanner.

FILES

/dev/escan

SEE ALSO

ioctl(2) in the *UNIX System V Programmer's Reference Manual.*

NAME

sxio - STREAM XIO device

DESCRIPTION

If an interactive process requires asynchronous I/O (XIO) requests and either STREAMS or sockets, *sxio* provides an interface to associate a selectable file descriptor with all XIO devices. A *select*(2B) or a *poll*(2) on the *sxio* file descriptor indicates that asynchronous XIO events have completed. Upon such an indication, the XIO system event flag mask must be read to process the completed XIO events. To clear the *sxio* file descriptor, *read*(2) using the *sxio* file descriptor until a 0 is returned.

FILES

/dev/sxio

SEE ALSO

intro(3A), select(2B) in the *CLIX Programmer's & User's Reference Manual.*
poll(2) in the *UNIX System V Programmer's Reference Manual.*

NOTES

The data returned from *sxio* is a list of the event flag numbers from asynchronous events in the order that they occurred.

NAME

tc - tape controller driver

DESCRIPTION

tc is the half-inch reel and quarter-inch cartridge tape driver. The special files **/dev/rmt/mt?[n]** refer to tape devices on the system Small Computer System Interface (SCSI) bus. The SCSI tape controller address is substituted for the ? in the special file name. The optional **n** denotes a no-rewind device.

The half-inch reel tape drive supports variable record sizes at variable densities. The quarter-inch cartridge tape drive only supports a fixed-block size of 512 bytes at fixed densities determined by QIC standards.

A typical special file to access a tape is

/dev/rmt/mt5n

where the SCSI controller address is 5 and the device is not rewound upon *close*(2). The format of the minor device mask for *tc* is shown below:

Minor Device Format:

7	6	5	4	3	2	1	0
IOP		UNIT		SCSI ID			REW

tc tape devices behave as described in *mtio*(7S). Exceptions exist for quarter-inch cartridge tape due to limitations with that type of device.

FILES

/dev/rmt/mt?[n]   special file for accessing tape driver

SEE ALSO

mtio(7S).
mt(1) in the *CLIX Programmer's & User's Reference Manual.*
ioctl(2) in the *UNIX System V Programmer's Reference Manual.*

NOTES

The largest possible transfer is 64K bytes if the buffer address is 1K byte aligned. Otherwise, the largest transfer is 62K bytes.

CAVEATS

The quarter-inch cartridge tape will only support *write*(2) operations after an initial tape load, rewind, forward-skip-to-EOT, write-file-mark, or a preceding write.

Since the quarter-inch cartridge tape supports fixed-block mode only, transfer counts must be multiples of 512 bytes and record sizes are not preserved. In addition, a partial write may occur just before the indication of end-of-tape (EOT). An EOT on write is indicated by return value of 0.

NAME

tcp - Transmission Control Protocol STREAMS device

DESCRIPTION

tcp is a "cloneable" (see *clone*(7)) STREAMS multiplexing driver that provides the services of the Transmission Control Protocol (TCP).

tcp communicates using the AT&T Transport Provider Interface (TPI). Adherence to TPI allows applications to interface with *tcp* using the AT&T Transport Layer Interface (TLI). *tcp* provides the TLI network service type T_COTS_ORD, a connection-oriented protocol service with orderly release. *tcp* should be linked above the *ip*(7S) STREAMS driver that has been bound to the TCP protocol (0x06) in order to provide the TCP Internet Protocol (IP) functionality of the Department of Defense (DoD) IP suite. *incd*(1M) performs this binding at boot time.

tcp uses TLI expedited data to implement TCP urgent data. *tcp* does not support the Transport Service Data Unit (TSDU) concept and will ignore the T_MORE flag in *t_snd*(3N).

The following address format is used as defined in **<sys/dod/inet.h>** and **<sys/dod/dod_ut.h>**:

```
typedef struct inet_addr {
        unchar uc[4];
} ina_t;

typedef struct dod_ut_addr {
        ina_t  inet;
        unchar port[2];
} dodaddr_t;
```

The following helpful macros are defined in **<sys/dod/dod_ut.h>** for converting the port between host byte order (least significant byte first) and network byte order (most significant byte first):

```
#define NET16_TO_HOST16(c)    (c[0] << 8 | c[1])
#define HOST16_TO_NET16(i, c) (c[0] = ((unchar)((i) >> 8))), \
                              (c[1] = ((unchar)((i))))
```

Each of the fields in the address format structures should be filled with values in network byte order. For example, the proper encoding of the Internet address 129.135.200.7 with port number 200 would be assigned to the structure as follows:

```
dodaddr_t     address;

address.inet.uc[0] = (unchar) 129;
address.inet.uc[1] = (unchar) 135;
address.inet.uc[2] = (unchar) 200;
address.inet.uc[3] = (unchar) 7;

HOST16_TO_NET16(200, address.port);
```

Bind requests inform *tcp* of the desired local address and port number to be associated with the stream. Bind requests contain either a *dod_ut_addr* structure or a zero length in the address specification.

Bind requests containing all zeros in the *inet* field indicate to *tcp* that any local host address is acceptable for connecting to remote hosts. Otherwise, the host address in the *inet* field must contain a valid local host address. Bind requests with all zeros in the *port* field tell *tcp* to find and allocate the first free port in the dynamic range for this stream. Otherwise, the *port* field indicates the port to be used. Bind requests with an address length of zero indicate the same information to *tcp* that a bind request with all zeros in the *inet* and *port* fields does.

If an existing stream is listening for incoming connect indications and another stream makes a bind request to listen on the same port, the request may fail. The failure will occur if the host address matches that of the existing stream or if either host address was specified as all zeros.

A maximum of one connect indication will be queued by a listening *tcp* stream.

tcp supports the Address Resolution Protocol (ARP) *ioctl*(2) requests described in *arp*(7B).

tcp supports options when establishing a connection. The local address that will be used throughout the lifetime of a connection is passed upstream from *tcp* as options in both the T_CONN_IND and T_CONN_CON primitives. The format of this address is one *dod_ut_addr* structure.

Because *tcp* will allow only one listening stream per TCP port, the presence of any listening stream, including one in a connected state, will disallow other streams from being bound to that port number for listening. A stream that has been bound as a listening stream can be marked as nonlistening in an option passed with the T_CONN_REQ primitive. The format of this option is one *tcp_creq_opt* structure defined in **<sys/dod/tcp.h>**. The *dont_listen* field in the structure should be set to nonzero if this stream is not to be used as a listening stream again.

FILES

/dev/tcp	special device file for TCP
<sys/dod/tcp.h>	definitions for *tcp* device
<sys/dod/tcph.h>	definitions for TCP protocol
<sys/dod/inet.h>	Internet address definitions
<sys/dod/dod_ut.h>	Internet utility and address definitions

SEE ALSO

arp(7B), icmp(7S), ip(7S), incd(1M), udp(7S).
clone(7) in the *UNIX System V Programmer's Reference Manual*.
Section (3N) in the *UNIX System V Programmer's Reference Manual*.
UNIX System V Network Programmer's Guide.

CAVEATS

incd(1M) is used at boot time to configure the STREAMS drivers and modules that implement network protocols, including *tcp*. Manual configuration is not usually necessary.

NAME

termio - general terminal interface

DESCRIPTION

All asynchronous communications ports use the same general interface, regardless of the hardware involved. The remainder of this section discusses the common features of this interface.

When a terminal file is opened, it normally causes the process to wait until a connection is established. In practice, user programs seldom open terminal files; they are opened by *getty*(1M) and become a user's standard input, output, and error files. The first terminal file opened by the process group leader of a terminal file not already associated with a process group becomes the control terminal for that process group. The control terminal plays a special role in handling quit and interrupt signals (discussed below). The control terminal is inherited by a child process during a *fork*(2). A process can break this association by changing its process group using *setpgrp*(2).

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

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

During input, erase and kill processing is normally performed. By default, the # character erases the last character typed, except that it will not erase beyond the beginning of the line. By default, the @ character kills (deletes) the entire input line and optionally outputs a newline character. Both characters operate on a key-stroke basis, independent from any backspacing or tabbing. Both the erase and kill characters may be entered literally by preceding them with the escape character (\). In this case, the escape character is not read. The erase and kill characters may be changed.

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

INTR (Rubout or ASCII DEL) generates an *interrupt* signal sent to all processes with the associated control terminal. Normally, each process is forced to terminate, but arrangements may be made to ignore the signal or to receive a trap to an agreed-on location (see *signal*(2)).

QUIT (<CONTROL>-L or ASCII FS) generates a *quit* signal. Its treatment is identical to the interrupt signal except that, unless a receiving process has made other arrangements, it will not only be terminated. However, a *core*(4) image file (called **core**) will be created in the current working directory.

SWTCH (<CONTROL>-Z or ASCII SUB) is used by the job control facility, *shl*, to change the current layer to the control layer.

SUSP (<CONTROL>-Z or ASCII SUB) generates a SIGTSTP signal sent to all processes with the associated control terminal. Normally, each process is forced to stop, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location (see *signal*(2)).

ERASE (#) erases the preceding character. It will not erase beyond the start of a line, as delimited by a NL, EOF, or EOL character.

KILL (@) deletes the entire line, as delimited by a NL, EOF, or EOL character.

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

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

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

EOL2 is an additional line delimiter.

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

START (<CONTROL>-Q or ASCII DC1) is used to resume output that has been suspended by a STOP character. While output is not suspended, START characters are ignored and not read. The start/stop characters cannot be changed or escaped.

The character values for INTR, QUIT, SWTCH, SUSP, ERASE, KILL, EOF, and EOL may be changed. The ERASE, KILL, and EOF characters may be escaped by a preceding \ character. In this case, no special function is performed.

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

When one or more characters is written, the characters are transmitted to the terminal as soon as previously-written characters finish typing. Input characters are echoed by putting them in the output queue as they arrive. If a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds its limit. When the queue has drained to a threshold, the program is resumed.

Several *ioctl*(2) system calls apply to terminal files. The primary calls use the following structure defined in **<termio.h>**:

```
#define   NCC              9
struct    termio {
          unsigned short   c_iflag;        /* input modes */
          unsigned short   c_oflag;        /* output modes */
          unsigned short   c_cflag;        /* control modes */
          unsigned short   c_lflag;        /* local modes */
          char             c_line;         /* line discipline */
          unsigned char    c_cc[NCC];      /* control chars */
};
```

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

0	VINTR	DEL
1	VQUIT	FS
2	VERASE	#
3	VKILL	@
4	VEOF	EOT
5	VEOL	NUL
6	reserved	
7	VSWTCH	
8	VSUSP	

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

IGNBRK	0000001	Ignore break condition.
BRKINT	0000002	Signal interrupt on break.
IGNPAR	0000004	Ignore characters with parity errors.
PARMRK	0000010	Mark parity errors.
INPCK	0000020	Enable input parity check.
ISTRIP	0000040	Strip character.
INLCR	0000100	Map NL to CR on input.
IGNCR	0000200	Ignore CR.
ICRNL	0000400	Map CR to NL on input.
IUCLC	0001000	Map uppercase to lowercase on input.
IXON	0002000	Enable start/stop output control.
IXANY	0004000	Enable any character to restart output.

IXOFF	0010000	Enable start/stop input control.

If IGNBRK is set, the break condition (a character framing error with data all zeros) is ignored (not put on the input queue and therefore not read by any process). Otherwise, if BRKINT is set, the break condition will generate an interrupt signal and flush both the input and output queues. If IGNPAR is set, characters with other framing and parity errors are ignored.

If PARMRK is set, a character with a framing or parity error not ignored is read as the three-character sequence: 0377, 0, *X*, where *X* is the data of the character received in error. To avoid ambiguity in this case, if ISTRIP is not set, a valid character of 0377 is read as 0377, 0377. If PARMRK is not set, a framing or parity error which is not ignored is read as the character NUL (0).

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

If ISTRIP is set, valid input characters are first stripped to seven bits. Otherwise, all eight bits are processed.

If INLCR is set, a received NL character is translated into a CR character. If IGNCR is set, a received CR character is ignored (not read). Otherwise, if ICRNL is set, a received CR character is translated into a NL character.

If IUCLC is set, a received uppercase alphabetic character is translated into the corresponding lowercase character.

If IXON is set, start/stop output control is enabled. A received STOP character will suspend output and a received START character will restart output. All start/stop characters are ignored and not read. If IXANY is set, any input character, will restart output that has been suspended.

If IXOFF is set, the system will transmit START/STOP characters when the input queue is nearly empty/full.

The initial input control value is all-bits-clear.

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

OPOST	0000001	Postprocess output.
OLCUC	0000002	Map lowercase to uppercase on output.
ONLCR	0000004	Map NL to CR-NL on output.
OCRNL	0000010	Map CR to NL on output.
ONOCR	0000020	No CR is output at column 0.
ONLRET	0000040	NL performs CR function.
OFILL	0000100	Use fill characters for delay.
OFDEL	0000200	Fill is DEL, else NUL.
NLDLY	0000400	Select newline delays:
NL0	0000000	
NL1	0000400	
CRDLY	0003000	Select carriage-return delays:
CR0	0000000	
CR1	0001000	

CR2	0002000	
CR3	0003000	
TABDLY	0014000	Select horizontal-tab delays:
TAB0	0000000	
TAB1	0004000	
TAB2	0010000	
TAB3	0014000	Expand tabs to spaces.
BSDLY	0020000	Select backspace delays:
BS0	0000000	
BS1	0020000	
VTDLY	0040000	Select vertical-tab delays:
VT0	0000000	
VT1	0040000	
FFDLY	0100000	Select form-feed delays:
FF0	0000000	
FF1	0100000	

If OPOST is set, output characters are postprocessed as indicated by the remaining flags. Otherwise, characters are transmitted without change.

If OLCUC is set, a lowercase alphabetic character is transmitted as the corresponding uppercase character. This function is often used with IUCLC.

If ONLCR is set, the NL character is transmitted as the CR-NL character pair. If OCRNL is set, the CR character is transmitted as the NL character. If ONOCR is set, no CR character is transmitted at column 0 (first position). If ONLRET is set, the NL character is assumed to perform the carriage-return function; the column pointer will be set to 0 and the delays specified for CR will be used. Otherwise the NL character is assumed to perform only the line-feed function; the column pointer will remain unchanged. The column pointer is also set to 0 if the CR character is actually transmitted.

The delay bits specify how long transmission stops to allow for mechanical or other movement when certain characters are sent to the terminal. In all cases a value of 0 indicates no delay. If OFILL is set, fill characters will be transmitted for delay instead of a timed delay. This is useful for high baud rate terminals that need only a minimal delay. If OFDEL is set, the fill character is DEL. Otherwise, it is NUL.

If a form-feed or vertical-tab delay is specified, it lasts for about two seconds.

Newline delay lasts about 0.10 seconds. If ONLRET is set, the carriage-return delays are used instead of the newline delays. If OFILL is set, two fill characters will be transmitted.

Carriage-return delay type 1 depends on the current column position; type 2 is about 0.10 seconds and type 3 is about 0.15 seconds. If OFILL is set, delay type 1 transmits two fill characters and type 2 transmits four fill characters.

Horizontal-tab delay type 1 depends on the current column position. Type 2 is about 0.10 seconds. Type 3 specifies that tabs are to be expanded into spaces. If OFILL is set, two fill characters will be transmitted for any delay.

Backspace delay lasts about 0.05 seconds. If OFILL is set, one fill character will be transmitted.

The actual delays depend on line speed and system load.

The initial output control value is all bits clear.

The *c_cflag* field describes the hardware control of the terminal:

CBAUD	0000017	Baud rate:
B0	0000000	Hang up
B50	0000001	50 baud
B75	0000002	75 baud
B110	0000003	110 baud
B134	0000004	134 baud
B150	0000005	150 baud
B200	0000006	200 baud
B300	0000007	300 baud
B600	0000010	600 baud
B1200	0000011	1200 baud
B1800	0000012	1800 baud
B2400	0000013	2400 baud
B4800	0000014	4800 baud
B9600	0000015	9600 baud
B19200	0000016	19200 baud
EXTA	0000016	External A
B38400	0000017	38400 baud
EXTB	0000017	External B
CSIZE	0000060	Character size:
CS5	0000000	5 bits
CS6	0000020	6 bits
CS7	0000040	7 bits
CS8	0000060	8 bits
CSTOPB	0000100	Send two stop bits. Otherwise, send one.
CREAD	0000200	Enable receiver.
PARENB	0000400	Enable parity.
PARODD	0001000	Use odd parity. Otherwise, use even.
HUPCL	0002000	Hang up on last close.
CLOCAL	0004000	Set local line. Otherwise, set dial-up.
RCV1EN	0010000	
XMT1EN	0020000	
LOBLK	0040000	Block layer output.

The CBAUD bits specify the baud rate. The zero baud rate, B0, hangs up the connection. If B0 is specified, the data-terminal-ready signal will not be asserted. Normally, this will disconnect the line. For any particular hardware, impossible speed changes are ignored.

The CSIZE bits specify the character size in bits for transmission and reception. This size does not include the parity bit, if any. If CSTOPB is set, two stop bits are used; otherwise, one stop bit is used. For example, at 110 baud, two stops bits are required.

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

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

If HUPCL is set, the line will be disconnected when the last process with the line open closes it or terminates. That is, the data-terminal-ready signal will not be asserted.

If CLOCAL is set, the line is assumed to be a local, direct connection with no modem control. Otherwise, modem control is assumed.

If LOBLK is set, the output of a job control layer will be blocked when it is not the current layer. Otherwise, the output generated by that layer will be multiplexed onto the current layer.

The initial hardware control value after open is B300, CS8, CREAD, HUPCL.

The *c_lflag* field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline (0) provides the following:

ISIG	0000001	Enable signals.
ICANON	0000002	Canonical input (erase and kill processing).
XCASE	0000004	Canonical upper/lower presentation.
ECHO	0000010	Enable echo.
ECHOE	0000020	Echo erase character as BS-SP-BS.
ECHOK	0000040	Echo NL after kill character.
ECHONL	0000100	Echo NL.
NOFLSH	0000200	Disable flush after interrupt or quit.
TOSTOP	0000400	SIGTTOU on background output.

If ISIG is set, each input character is checked against the special control characters INTR, SWTCH, SUSP, and QUIT. If an input character matches one of these control characters, the function associated with that character is performed. If ISIG is not set, no checking is performed. Thus these special input functions are possible only if ISIG is set. These functions may be disabled individually by changing the value of the control character to an unlikely or impossible value (such as 0377).

If ICANON is set, canonical processing is enabled. This enables the erase and kill edit functions and the assembly of input characters into lines delimited by NL, EOF, and EOL. If ICANON is not set, read requests are satisfied directly from the input queue. A read will not be satisfied until at least MIN characters have been received or the timeout value TIME has expired between characters. This allows fast bursts of input to be read efficiently while still allowing single character input. The MIN and TIME values are stored in the position for the EOF and EOL characters, respectively. The time value represents tenths of seconds.

If XCASE and ICANON are set, an uppercase letter is accepted on input when it is preceded by a \ character and is output preceded by a \ character. In this mode, the following escape sequences are generated on output and accepted at input:

<table>
<tr><td>for:</td><td>use:</td></tr>
<tr><td>*</td><td>*</td></tr>
<tr><td>|</td><td>\!</td></tr>
<tr><td>~</td><td>\^</td></tr>
<tr><td>{</td><td>\(</td></tr>
<tr><td>}</td><td>\)</td></tr>
<tr><td>\</td><td>\\</td></tr>
</table>

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

If ECHO is set, characters are echoed as received.

When ICANON is set, the following echo functions are possible. If ECHO and ECHOE are set, the erase character is echoed as ASCII BS SP BS, which will clear the last character from a CRT screen. If ECHOE is set and ECHO is not set, the erase character is echoed as ASCII SP BS. If ECHOK is set, the NL character will be echoed after the kill character to emphasize that the line will be deleted. Note that an escape character preceding the erase or kill character removes any special function. If ECHONL is set, the NL character will be echoed even if ECHO is not set. This is useful for terminals set to local echo (known as half duplex). Unless escaped, the EOF character is not echoed. EOT being the default EOF character prevents terminals that respond to EOT from hanging up.

If NOFLSH is set, the normal flush of the input and output queues associated with the quit, switch, and interrupt characters will not be performed.

If TOSTOP is set, background processes attempting to write to a controlling terminal will generate a SIGTTOU signal sent to all processes with the associated control terminal.

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

The primary *ioctl*(2) system calls have the following form:

> ioctl(*fildes*, *request*, *arg*)
> struct termio **arg*;

The *requests* using this form are as follows:

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

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

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

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

Additional *ioctl*(2) calls have the following form:

ioctl(*fildes*, *request*, *arg*)
int *arg*;

The *requests* using this form are as follows:

TCSBRK Wait for the output to drain. If *arg* is 0, send a break (zero bits for 0.25 seconds).

TCXONC Start/stop control. If *arg* is 0, suspend output; if 1, restart suspended output.

TCFLSH If *arg* is 0, flush the input queue; if 1, flush the output queue; and if 2, flush both the input and output queues.

Additional *ioctl*(2) calls have the following form:

ioctl(*fildes*, *request*, *arg*)
int **arg*;

The *requests* using this form are as follows:

TCGPGRP Get the distinguished process group ID associated with the terminal and store it in the integer location referenced by *arg*.

TCSPGRP Set the distinguished process group ID associated with the terminal from the integer location referenced by *arg*. The change is immediate.

FILES

/dev/tty*

SEE ALSO

stty(1), setpgrp(2), signal(2) in the *CLIX Programmer's & User's Reference Manual*.
fork(2), ioctl(2) in the *UNIX System V Programmer's Reference Manual*.

NAME

tidcl - STREAMS DoD UDP driver

DESCRIPTION

tidcl is a STREAMS driver that supports the Transport Layer Interface (TLI). It provides the network service type T_CLTS, which is a connectionless (datagram) protocol service.

NOTES

/dev/tidcl is now supported by modifications to **/dev/udp**, which is type T_CLTS. **/dev/udp** uses a different address format than *tidcl* and also supports options. Existing programs that use *tidcl* will still run without modification. The new *udp*(7S) device should be used for all newly created applications requiring the Department of Defense (DoD) User Datagram Protocol (UDP). Support for *tidcl* is not guaranteed in future CLIX releases.

FILES

/dev/tidcl special device file for *tidcl*

SEE ALSO

udp(7S).
Section (3N) in the *CLIX Programmer's & User's Reference Manual.*

NAME

tidco - STREAMS DoD TCP driver

DESCRIPTION

tidco is a STREAMS driver that supports the Transport Layer Interface (TLI). It provides the network service type T_COTS, which is a connection-oriented protocol service with no orderly release.

NOTES

/dev/tidco is now supported by modifications to **/dev/tcp**, which is type T_COTS_ORD. **/dev/tcp** uses a different address format than *tidco* and also supports options. Existing programs that use *tidco* will still run without modification. The new *tcp*(7S) device should be used for all newly created applications requiring the Department of Defense (DoD) Transmission Control Protocol (TCP). Support for *tidco* is not guaranteed in future CLIX releases.

FILES

/dev/tidco special device file for *tidco*

SEE ALSO

tcp(7S).
Section (3N) in the *CLIX Programmer's & User's Reference Manual.*

NAME

tixco - STREAMS XNS SPP driver

DESCRIPTION

tixco is a streams driver that supports the Transport Layer Interface (TLI). It provides the network service type T_COTS, which is a connection-oriented protocol service with no orderly release.

NOTE

/dev/tixco is linked to **/dev/xs**, which is of type T_COTS_ORD, a connection-oriented protocol with orderly release. The address format of *xs*(7S) is the same as that of *tixco*, and existing programs that use *tixco* will still run without any modification. The new *xs*(7S) device should be used for all newly created applications requiring the Xerox Network Services (XNS) Sequenced Packet Protocol (SPP). Support for *tixco* is not guaranteed in future CLIX releases.

FILES

/dev/tixco   special device file for *tixco*

SEE ALSO

xs(7S).
Section (3N) in the *CLIX Programmer's & User's Reference Manual.*

NAME

tren - DoD Trailer decapsulation STREAMS module

DESCRIPTION

tren is the Department of Defense (DoD) trailer decapsulation STREAMS module created to improve the performance of the Transmission Control Protocol/Internet Protocol (TCP/IP) on some machines. A frame format that placed the transport layer data immediately after the Ethernet header, with the transport headers following the data was created. Early versions of this trailer mechanism did not provide a way to turn trailer encapsulation on and off. To communicate with a system with trailer encapsulation on, it is necessary on a CLIX machine to push the *tren* module between the *ip*(7S) device and the *et*(7S) Logical Link Interface (LLI) device to decapsulate these frames. Doing so puts the transport layer headers immediately after the Ethernet header followed by the transport layer data.

To set up a stream to be linked below *ip*(7S) and that will decapsulate incoming trailer encapsulated frames, the LLI device must be opened for the desired physical network (**/dev/et0**) and the device must be bound to the trailer Service Access Point (SAP) TRLR_SAP (0x1001). Then the *tren* streams module must be pushed onto the stream. Finally, the device **/dev/ip** must be opened and the *tren* module linked below the *ip*(7S) stream.

The trailer module may be pushed only onto a stream terminated by an LLI device that is bound to the trailer SAP.

Once the *tren* module has been pushed onto a stream, all messages received on the module's write side will be transparently passed on to the module's downstream neighbor. All messages received on the module's read side will be transparently passed on to the module's upstream neighbor as well, with the exception of M_PROTO messages carrying DL_UNITDATA_IND primitives. These messages contain trailer encapsulated frames that will be decapsulated to look like ordinary *ip*(7S) frames and passed upstream.

FILES

<sys/lihdr.h>	LLI header file

SEE ALSO

et(7S), incd(1M), ip(7S), streamio(7).
ioctl(2) in the *CLIX Programmer's & User's Reference Manual.*

CAVEATS

incd(1M) is used at boot time to configure the STREAMS drivers and modules that implement network protocols, including *tren*. Manual configuration is not usually necessary.

NAME

uco - STREAMS UNIX domain connection-oriented driver

DESCRIPTION

uco is a STREAMS driver which supports the Transport Layer Interface (TLI). It provides the network service of type T_COTS which is a connection-oriented protocol service with no orderly release. Any outstanding data is flushed on close or disconnect.

This device supports the transfer of arbitrarily large data packets as well as expedited data packets. This device does not support the use TLI options.

The following address format defined in **<sys/uco.h>** is used to bind and connect devices.

```
struct uco_addr {
        ushort port;
};
```

An address length of 0 in the *t_bind* structure specifies that the port number is chosen by the *uco* driver.

FILES

/dev/tiuco

SEE ALSO

Section (3N) of the *CLIX Programmer's & User's Reference Manual.*
Section (3N) of the *UNIX System V Programmer's Reference Manual.*
UNIX System V Network Programmer's Guide.

NAME

udp - User Datagram Protocol

DESCRIPTION

udp is a "cloneable" (see *clone*(7)) STREAMS multiplexing driver that provides the services of the User Datagram Protocol (UDP).

upd communicates using the AT&T Transport Provider Interface (TPI). Adherence to TPI allows applications to interface with *udp* using the AT&T Transport Layer Interface (TLI). *udp* provides the TLI network service type T_CLTS, a connectionless (datagram) protocol service. *udp* should be linked above the *ip*(7S) STREAMS driver that has been bound to the UDP protocol (0x11) in order to provide the UDP/Internet Protocol (IP) functionality of the DARPA Internet Protocol suite. *incd*(1M) performs this binding at boot time.

The following address format is used as defined in **<sys/dod/inet.h>** and **<sys/dod/dod_ut.h>**:

```
typedef struct inet_addr {
        unchar uc[4];
} ina_t;

typedef struct dod_ut_addr {
        ina_t  inet;
        unchar port[2];
} dodaddr_t;
```

The following helpful macros are defined in **<sys/dod/dod_ut.h>** for converting the port between host byte order (least significant byte first) and network byte order (most significant byte first):

```
#define NET16_TO_HOST16(c)    (c[0] << 8 | c[1])
#define HOST16_TO_NET16(i, c) (c[0] = ((unchar)((i) >> 8))), \
                              (c[1] = ((unchar)((i))))
```

Each of the fields in the address format structures should be filled with values in network byte order. For example, the proper encoding of the Internet address 129.135.200.7 with port number 200 would be assigned to the structure as follows:

```
dodaddr_t      address;

address.inet.uc[0] = (unchar) 129;
address.inet.uc[1] = (unchar) 135;
address.inet.uc[2] = (unchar) 200;
address.inet.uc[3] = (unchar) 7;

HOST16_TO_NET16(200, address.port);
```

When a *t_bind*(3N) is requested, an address length of zero in the *t_bind*(3N) structure specifies that a port number be chosen by the driver for all local endpoints. If a specific port needs to be bound, the address length should be set to DOD_UT_ADDR_SZ and the port field should contain the port number to be bound. The maximum number of outstanding connect indications should be set to zero. If no address is specified or the Internet address zero is specified, all local interfaces will be bound and an all zero Internet addresses will be returned as the bound addresses.

When data is sent (with *t_sndudata*(3N)), the destination address length in the *t_unitdata* structure should be set to DOD_UT_ADDR_SZ and the entire remote address, including the Internet address and port number, should be specified. Data to be filled in the Internet Protocol header including the TYPE OF SERVICE, TIME TO LIVE, IP OPTIONS, and source Internet address may be specified with the *opt* field in the *t_unitdata* structure using the format of the following structure defined in **<sys/dod/udp.h>**:

```
struct udp_udopt {
        ina_t       addr;
        unchar      iptos;
        unchar      ipttl;
        unchar      ipoptions[IPH_MAX_OPT_SZ];
        dodaddr_t;
}
```

When data is received (with *t_rcvudata*(3N)), the destination Internet address, TYPE OF SERVICE, and OPTIONS from the Internet Protocol header will be returned with the *opt* field in the *t_unitdata* structure using the format of the *udp_udopt* structure.

Internet Control Message Protocol (ICMP) error messages pertaining to datagrams sent with *udp*(7S) may be retrieved through the *t_rcvuderr*(3N) call. *udp*(7S) supports the Address Resolution Protocol *ioctl*(2) requests described in *arp*(7B).

FILES

/dev/udp	special device file for UDP
<sys/dod/inet.h>	Internet address definitions
<sys/dod/dod_ut.h>	Internet utility and address definitions
<sys/dod/udp.h>	definitions for *udp* device

SEE ALSO

arp(7B), icmp(7S), ip(7S), tcp(7S), incd(1M).
Section (3N) in the *CLIX Programmer's & User's Reference Manual.*
clone(7) in the *UNIX System V System Administrator's Reference Manual.*

CAVEATS

incd(1M) is used at boot time to configure the STREAMS drivers and modules that implement network protocols, including *udp*. Manual configuration is not usually necessary.

NAME

xnsxt - Intergraph XT STREAMS terminal driver

DESCRIPTION

xnsxt allows remote logins using the Intergraph XT protocol. A connection is made from a remote machine by the *visit*(1) program. The listener process, *xxt_listener*(1M), on the local machine establishes the connection. When the connection is established, the listener transfers the connection to the *xnsxt* driver, pushes an *ldterm*(7S) STREAMS module, and starts a *getty*(1M) on a port assigned by the driver. When *getty*(1M) or any program that *getty*(1M) *execs*(2) terminates, the listener queries the *xnsxt* driver to determine if the *getty*(1M) should be restarted.

FILES

/dev/ttn??	network terminals
/dev/xt	control device for Intergraph XT protocol

SEE ALSO

xxt_listener(1M), termio(7S), ldterm(7S).
visit(1) in the *CLIX Programmer's & User's Reference Manual.*

NAME

xpe - STREAMS XNS PEP driver

DESCRIPTION

xpe is a STREAMS driver that provides the services of the Packet Exchange Protocol (PEP) to applications and other protocols on the host machine. PEP is the transaction-based connectionless transport layer protocol of the Xerox Network System (XNS) protocol suite.

clone(7) opens may be performed on the *xpe* device to find the first available free minor device.

xpe communicates on its upper streams using the AT&T Transport Provider Interface (TPI) providing support for the AT&T Transport Layer Interface (TLI) to applications. *xpe* provides the T_CLTS connectionless protocol service (as specified by TPI/TLI) to its upstream clients.

The PEP protocol defines a protocol that exchanges packets between a client socket and a server socket. The client side generates PEP requests and sends them to the server. The server side receives client requests from the network, does some processing, and returns PEP responses to the clients. TLI makes no such client/server distinction for its connectionless devices. To provide for the distinction, *xpe* may be put into client or server mode when the bind request is made, as described below.

In client mode, unitdata requests will be sent downstream to *xpe* and a corresponding PEP request will be sent. *xpe* will then wait for the PEP response to return from the server, possibly retransmitting the PEP request if no response is seen for the specified retransmit timeout. If a matching PEP response (or responses) arrives, a unitdata indication will be sent upstream by *xpe*. If no matching PEP response arrives within the specified timeout, a unitdata error indication is sent upstream by *xpe*. In client mode, PEP requests sent to the socket bound by the client will be silently ignored.

In server mode, *xpe* waits for incoming PEP requests sent to the local socket and sends corresponding unitdata indications upstream. To respond to the PEP request, the server sends unitdata requests downstream to *xpe*.

The address format used in the bind, unitdata request, and unitdata indication operations is an array of 12 bytes. The network number is contained in the first four bytes of the array, the host address is contained in the next six bytes of the array, and the socket number is contained in the last two bytes of the array. Each of the three numbers (network, host, and socket) is filled in with network order (most significant byte first.)

For example, to fill in an address structure with the network address 0x000134ab, host address 08-00-36-ab-cd-03, and socket number 0x0045 the following C code is used:

```
char    addr[12];
addr[0] = 0x00;
addr[1] = 0x01;
addr[2] = 0x34;
```

```
addr[3] = 0xAB;
addr[4] = 0x08;
addr[5] = 0x00;
addr[6] = 0x36;
addr[7] = 0xAB;
addr[8] = 0xCD;
addr[9] = 0x03;
addr[10] = 0x00;
addr[11] = 0x45;
```

Only the socket part of the address is meaningful in bind requests, telling *xpe* which socket to bind. On return from a successful bind, the entire bound address will be placed in the *ret* address parameter buffer. The socket number returned may be different than the one requested if that socket is already in use elsewhere in the system.

When binding, the *qlen* value of the *req* structure determines whether the stream will be used for PEP client or PEP server processing. A *qlen* of zero indicates a client; nonzero indicates a server.

xpe supports TLI options associated with unitdata requests and unitdata indications. The options format is the following *xpeopts_s* structure defined in **<sys/xns/xns.h>**:

```
typedef unsigned char  u8;
typedef struct xpeopts_s {
        u8      xpeo_client_type[2];
        char    xpeo_retries;
        char    xpeo_seconds_per_retry;
        u8      xpeo_id[4];
        u8      xpeo_pass_xropts;
        u8      xpeo_wants_multiple_replies;
} XPE_OPTS;
```

The *xpeo_client_type* field contains the PEP client type to be sent by unitdata requests or that was received in unitdata indications.

The *xpeo_retries* field is always 0 in unitdata indications. It indicates the number of retransmissions that should be attempted from client *xpe* streams. Packets sent from *xpe* server streams are not retransmitted. For clients, if this value is set to zero in a unitdata request, the default retry count of 10 is used.

The *xpeo_seconds_per_retry* field is always 0 in unitdata indications. For client unitdata requests (PEP requests) this value indicates the number of seconds between retries. If this value is set to 0, the first retry will occur after two seconds. Subsequent retries will be based on a round-trip time estimate obtained from the downstream *xr*(7S) module for the specified destination address.

The *xpeo_id* field contains the PEP ID of the received packet in unitdata indications. For *xpe* server streams, this field indicates the value to use for the

PEP ID of the response being sent (allowing servers to match response PEP IDs to the client's request PEP IDs). For *xpe* client streams, this field is ignored, and *xpe* generates a unique ID for each client packet.

The *xpeo_pass_xropts* field is always 0 in unitdata indications. In unitdata requests, if this field is nonzero, an *xr_opts* structure should immediately follow the *xpe_opts* structure in the unitdata request options buffer. This option is used only to support expanding rings.

The *xpeo_wants_multiple_replies* field is always 0 in unitdata indications. For *xpe* client streams, nonzero *xpeo_wants_multiple_replies* values indicate that all replies (not just the first one) to a client broadcast (or multicast) request should be received and passed upstream in unitdata indications.

FILES

/dev/xpe	special device file for *xpe*
<sys/xns/xns.h>	XNS header file
<sys/xns/common.h>	XNS common definitions header file
<sys/xns/mi.h>	XNS miscellaneous header file
<sys/lihdr.h>	Logical Link Interface header file

SEE ALSO

et(7S), xr(7S), xs(7S), clone(7), incd(1M).
Section (3N) in the *CLIX Programmer's & User's Reference Manual.*

CAVEATS

On CLIX systems, *incd*(1M) is usually responsible for creating the network configuration of streams drivers/modules/multiplexors, including *xpe*. Manual configuration is necessary only in rare circumstances.

NAME

xr - STREAMS IDP and RIP multiplexor

DESCRIPTION

xr is a STREAMS driver that provides the services of the Internet Datagram Protocol (IDP) to applications and other protocols on the host machine. It also automatically updates the XNS routing tables maintained in the kernel by listening to XNS Routing Information Protocol (RIP) network traffic. IDP is the connectionless network layer protocol of the Xerox Network System (XNS) Protocol suite.

clone(7) opens may be performed on the *xr* device to find the first available free minor device.

xr communicates on its upper streams using the AT&T Transport Provider Interface (TPI), providing support for the AT&T Transport Layer Interface (TLI) to applications. *xr* provides the T_CLTS connectionless protocol service (as specified by TPI/TLI) to its upstream clients. *xr* communicates on its lower streams using the AT&T Logical Link Interface (LLI) expecting that its lower streams are bound to the Service Access Point (SAP) XNS_SAP (0x600) before they are linked.

The address format used in the bind, unit datagram (unitdata) request, and unitdata indication operations is an array of 12 bytes. The network number is contained in the first four bytes of the array, the host address is contained in the next six bytes of the array, and the socket number is contained in the last two bytes of the array. Each of the three numbers (network, host, and socket) is filled in with network order (most significant byte first.)

For example, to fill in an address structure with the network address **0x000134ab**, host address **08-00-36-ab-cd-03**, and socket number **0x0045**, the following C code is used:

```
char     addr[12];
addr[0] = 0x00;
addr[1] = 0x01;
addr[2] = 0x34;
addr[3] = 0xAB;
addr[4] = 0x08;
addr[5] = 0x00;
addr[6] = 0x36;
addr[7] = 0xAB;
addr[8] = 0xCD;
addr[9] = 0x03;
addr[10] = 0x00;
addr[11] = 0x45;
```

Only the socket part of the address is meaningful in bind requests, telling *xr* which socket to bind. On return from a successful bind, the entire bound address will be placed in the *ret* address parameter buffer. The socket number returned may be different than the one requested if that socket is already in use elsewhere in the system.

xr supports TLI options associated with unitdata requests. The options format is the following *xropts_s* structure defined in <sys/xns/xns.h>:

```
typedef unsigned char  u8;
typedef struct xropts_s
{
        u8      xro_radius;     /* Send to nets radius hops away */
        char    xro_pad[3];     /* Force 4 byte alignment */
} XR_OPTS;
```

The *xro_radius* field contains the number of hops away to send a broadcast or multicast datagram. *xr* sends the packet to all networks that are *radius* hops away. This parameter is used to perform expanding rings.

FILES

/dev/xr	special device for *xr*
<sys/xns/xns.h>	XNS header file
<sys/xns/common.h>	XNS common definitions header file
<sys/xns/mi.h>	XNS miscellaneous header file

SEE ALSO

xpe(7S), xs(7S), clone(7), incd(1M).
Section (3N) in the *CLIX Programmer's & User's Reference Manual.*

CAVEATS

On CLIX systems, *incd*(1M) is usually responsible for creating the network configuration of streams drivers/modules/multiplexors, including *xr*. Manual configuration is necessary only in rare circumstances.

NAME

xs - STREAMS XNS SPP driver

DESCRIPTION

xs is a STREAMS driver that provides the services of the Sequenced Packet Protocol (SPP) to applications and other protocols on the host machine. SPP is the connection-oriented transport layer protocol of the Xerox Network System (XNS) Protocol suite.

clone(7) opens may be performed on the *xs* device to find the first available free minor device.

xs communicates on its upper streams using the AT&T Transport Provider Interface (TPI), providing support for the AT&T Transport Layer Interface (TLI) to applications. *xs* provides the T_COTS_ORD connection-oriented protocol service with orderly release (as specified by TPI/TLI) to its upstream clients.

xs supports the TLI expedited data concept and maps it to the SPP attention function. Only one-byte expedited data requests are allowed.

xs supports the Transport Service Data Unit (TSDU) concept and maps it to the SPP End Of Message (EOM) function.

The SPP header carries a one-byte field called the *datastream type*. The *datastream type* may change from one TSDU to another TSDU, so it needs to be sent to *xs* with each TLI data request. TLI makes no provisions for such a function, so the CLIX TLI routines were modified slightly from the AT&T standard to accommodate it. The *datastream type* associated with each data request is sent to *xs* in the *flags* field of the data request. The low-order bit of the *flags* field is used to indicate the T_MORE function. The byte of *datastream type* to use for the data will be sent to *xs* in bits 8-15 of the *flags* field. For example, to send down no T_MORE flag and a *datastream type* of 0x23, the following C code is used:

```
int     flags;

flags = (0x23 << 8);
t_snd(fd, buf, nbytes, flags);
```

datastream types 254 (0xFE) and 255 (0xFF) are used by *xs* to perform the orderly release three-way handshake and should not be specified as *datastream types* of normal data by applications.

xs does not currently support any TLI options.

The address format used in the bind and connect operations is an array of 12 characters. The network number is contained in the first four bytes of the array, the host address is contained in the next six bytes of the array, and the socket number is contained in the last two bytes of the array. Each of the three numbers (network, host, and socket) is filled in with network order (most significant byte first.) For example, to fill in an address structure with the network address **0x000134ab**, host address **08-00-36-ab-cd-03**, and socket number **0x0045** use the following C code:

```
char    addr[12];
addr[0] = 0x00;
addr[1] = 0x01;
addr[2] = 0x34;
addr[3] = 0xAB;
addr[4] = 0x08;
addr[5] = 0x00;
addr[6] = 0x36;
addr[7] = 0xAB;
addr[8] = 0xCD;
addr[9] = 0x03;
addr[10] = 0x00;
addr[11] = 0x45;
```

Only the socket part of the address is meaningful in bind requests, telling *xs* which socket to bind. On return from a successful bind, the entire bound address will be placed in the *ret* address parameter buffer. The socket number returned may be different than the one requested if that socket is already in use elsewhere in the system.

The maximum number of connect indications that will be queued by listening *xs* streams is 5.

FILES

/dev/xs	special device file for *xs*
<sys/xns/xns.h>	XNS header file
<sys/xns/common.h>	XNS common definitions header file
<sys/xns/mi.h>	XNS miscellaneous header file

SEE ALSO

xr(7S), xpe(7S), clone(7), incd(1M).
Section (3N) in the *CLIX Programmer's & User's Reference Manual.*

CAVEATS

On CLIX systems, *incd*(1M) is usually responsible for creating the network configuration of streams drivers/modules/multiplexors, including *xs*. Manual configuration is necessary only in rare circumstances.

NAME

xyl - multi-channel serial interface driver

DESCRIPTION

xyl is a serial interface driver used to access the Xylogics 16-channel RS232 serial card. Each port supports transmit data (TXD), receive data (RXD), request-to-send (RTS), clear-to-send (CTS), data terminal ready (DTR), and data set ready (DSR).

The *xyl* driver supports the Xylogics multi-channel serial ports as described in *termio*(7S). The number of available ports depends on the platform and system configuration.

FILES

/dev/ttz??

SEE ALSO

termio(7S).

NOTES

Any *open*(2) will return immediately allowing a process to perform an *ioctl*(2) which may specify CLOCAL. If CLOCAL is not set before the first *read*(2) or *write*(2), the operation waits for a carrier indicated by the DSR signal. If CLOCAL is set, the state of DSR is ignored.

WARNINGS

The lack of DCD may require special jumpering or special cables for some modems.

7B

NAME

intro - introduction to BSD networking facilities

SYNOPSIS

#include <sys/socket.h>
#include <net/route.h>
#include <net/if.h>

DESCRIPTION

This section briefly describes the Berkeley Software Distribution (BSD) networking facilities available in the system. All network protocols are associated with a specific *protocol family*. A protocol family provides basic services to the protocol implementation to allow it to function within a specific network environment. These services may include packet fragmentation and reassembly, routing, addressing, and basic transport. A protocol family may support multiple methods of addressing, though the current protocol implementations do not. A protocol family is normally comprised of a number of protocols, one per *socket*(2B) type. It is not required that a protocol family support all socket types. A protocol family may contain multiple protocols supporting the same socket abstraction.

A protocol supports one of the socket abstractions detailed in *socket*(2B). A specific protocol may be accessed either by creating a socket of the appropriate type and protocol family, or by requesting the protocol explicitly when creating a socket. Protocols normally accept only one type of address format, usually determined by the addressing structure inherent in the design of the protocol family/network architecture. Certain semantics of the basic socket abstractions are protocol specific. All protocols are expected to support the basic model for their particular socket type, but may, in addition, provide nonstandard facilities or extensions to a mechanism. For example, a protocol supporting the *sock_stream* abstraction may allow more than one byte of out-of-band data to be transmitted per out-of-band message.

A network interface is similar to a device interface. Network interfaces comprise the lowest layer of the networking subsystem, interacting with the actual transport hardware. An interface may support one or more protocol families and/or address formats.

Protocols

The system currently supports the Defense Advanced Research Project Agency (DARPA) Internet protocols and the Xerox Network Systems (XNS) protocols. Consult the appropriate manual pages in this section for more information regarding the support for each protocol family.

Addressing

Associated with each protocol family is an address format. The following address formats are used by the system (and additional formats are defined for possible future implementation):

```
#define AF_UNIX     1     /* local to host (pipes, portals) */
#define AF_INET     2     /* internetwork: UDP, TCP, etc. */
```

```
#define AF_NS        6        /* Xerox NS protocols */
```

Interfaces

Each network interface in a system corresponds to a path through which messages may be sent and received. A network interface usually has a hardware device associated with it.

SEE ALSO

socket(2B) in the *CLIX Programmer's & User's Reference Manual.*

NAME

arp - Address Resolution protocol

DESCRIPTION

arp is a STREAMS module that implements the Address Resolution Protocol (ARP) (see "An Ethernet Address Resolution Protocol," RFC826, Dave Plummer, Network Information Center, SRI). ARP is a protocol used to dynamically map between Defense Advanced Research Projects Agency (DARPA) Internet addresses and Ethernet addresses. It is not specific to Internet protocols or to Ethernet, but this implementation currently supports only that combination.

The *arp* module must be inserted with the I_PUSH *ioctl*(2) onto a stream terminated by an Ethernet device driver that conforms to the AT&T Logical Link Interface specifications (such as **/dev/et0**). The Ethernet device driver must be bound to the ARP Protocol (0x806). *incd*(1M) performes this binding at boot time.

To facilitate communications with systems that do not access ARP directly, STREAMS *ioctl*(2) requests are provided to add and delete entries in the Internet-to-Ethernet address tables. Two formats for the STREAMS *ioctl*(2) requests are supported; a local format as well as the standard Berkeley format. These STREAMS *ioctl*(2) requests may be applied to any open file descriptor referring to a STREAMS device that provides support for *arp* *ioctl*(2) requests. Several devices that support these requests are **/dev/ip**, **/dev/udp**, and **/dev/tcp**. The format of a STREAMS *ioctl*(2) request is as follows:

```
#include <sys/stropts.h>
#include <sys/dod/ar.h>
#include <net/dod/arp.h>

struct strioctl         ic;

ioctl (fd, I_STR, (caddr_t) &ic);

struct strioctl {
        int     ic_cmd;         /* command */
        int     ic_timout;      /* timeout value */
        int     ic_len;         /* length of data */
        char    *ic_dp;         /* pointer to data */
};
```

Ic_cmd should be set to one of the following commands as defined in **<sys/dod/arp.h>**:

I_ARPADD	Add an ARP entry to the local cache.
I_ARPDEL	Delete an ARP entry from the local cache.
I_ARPLOC	Retrieve an ARP entry from the local cache.
I_ARPTRNS	Retrieve an ARP entry by first looking in the local cache; if a translation is not found in the local cache, the net is queried for a translation.

I_ARPREQ	Query the net for a translation.
I_ARPTBL	Obtain all entries in the local ARP cache.

Ic_timout should be set to zero for all commands listed above. The timeout for network responses is set automatically to five seconds.

For I_ARPTBL, *ic_len* should reflect the size (in bytes) of the buffer pointed to by *ic_dp*. For the remainder of the commands, *ic_len* should be set to ARP_IOC_SZ as defined in **<sys/dod/ar.h>**.

The entries in the ARP cache have the following format as defined in **<sys/dod/arp.h>**:

```
struct arp_ioc {
        unchar inet[INET_ALEN];          /* Internet address */
        unchar enet[ENET_ALEN];          /* Ethernet address */
        ushort flags;                    /* arp flags */
};
```

For I_ARPTBL, *ic_dp* should contain a pointer to a buffer large enough to hold all entries in the *arp* cache. If the buffer is large enough to hold all of the entries, the value returned by *ioctl*(2) will be zero. Otherwise, the value returned will be the number of entries in the *arp* cache. For the remainder of the commands listed above, *ic_dp* should be a pointer to an *arp_ioc* structure. The addresses in *inet* and *enet* should be stored in network byte order (most significant byte first) and are returned in this format. For I_ARPADD, *flags* should be set to one of the following values as defined in **<sys/dod/ar.h>**:

AR_LOCAL_FL	translation for local interface
AR_PERM_FL	permanent entry
AR_PUBL_FL	reply to ARP requests from the network

For I_ARPDEL, *flags* must be set to AR_PERM_FL in order to successfully delete a permanent entry. For I_ARPLOC, I_ARPTRNS and I_ARPREQ, *flags* should be set to one of the following values as defined in **<sys/dod/ar.h>**:

INET	Retrieve the Ethernet address translation for the given *inet* address.
ENET	Retrieve the Internet address translation for the given *enet* address.

In addition to the above STREAMS *ioctl*(2) requests, support is provided for the following *ioctl*(2) requests to any *socket*(2B) descriptor:

```
struct arpreq {
        struct  sockaddr arp_pa;      /* protocol address */
        struct  sockaddr arp_ha;      /* hardware address */
        int     arp_flags;            /* flags */
} arpreq;
```

ioctl (sd, *cmd*, &arpreq);

The following *cmds* are vaild:

SIOCSARP	Set an ARP entry.
SIOCGARP	Get an ARP entry.
SIOCDARP	Delete an ARP entry.
SIOCXARP	Query the local network for an ARP entry.

The address format for *arp_pa* and *arp_ha* are described in *inet*(7B). The address family specified in *arp_pa* must be AF_INET and AF_UNSPEC in *arp_ha*.

Flags may be one of the following as defined in **<net/if_arp.h>**:

ATF_COM	completed entry
ATF_PERM	permanent entry

ATF_PERM is the only flag that can be written.

Only the super-user may add or delete entries in the ARP cache. If an *arp ioctl*(2) fails, *errno* will be set to one of the following:

[EINVAL]	A command other than I_ARPTBL was issued without a valid pointer in the *ic_dp* field.
[EPERM]	The effective user ID is not super-user.
[EAGAIN]	STREAMS resources are insufficient.
[EACCES]	Permission to perform the requested action is denied.
[ENOENT]	A translation was not found in the local ARP cache.
[ETIME]	The timer for the *arp* request has expired.

FILES

<sys/dod/inet.h>	special device file for UDP
<sys/dod/dod_ut.h>	Internet address definitions
<sys/dod/ar.h>	
<sys/dod/arp.h>	

SEE ALSO

inet(7B).
ioctl(2) in the *UNIX System V Programmer's Reference Manual*.

CAVEATS

incd(1M) is normally executed at boot time to configure the STREAMS drivers and modules that implement network protocols, including *arp*.

NAME

inet - Internet protocol family

SYNOPSIS

#include <sys/types.h>
#include <netinet/in.h>

DESCRIPTION

The Internet protocol family is a collection of protocols layered on top of the Internet Protocol (IP) transport layer using the Internet address format. The Internet family provides protocol support for the *sock_stream* and *sock_dgram* socket types.

Internet addresses are four-byte quantities, stored in network standard format (on the CLIPPER™ these are word and byte reversed). The include file **<netinet/in.h>** defines this address as a discriminated union.

Sockets bound to the Internet protocol family use the following addressing structure:

```
struct sockaddr_in {
        short       sin_family;
        u_short     sin_port;
        struct      in_addr sin_addr;
        char        sin_zero[8];
};
```

Sockets may be created with the local address INADDR_ANY to effect "wildcard" matching on incoming messages. The address in a *connect*(2B) or *sendto*(2B) call may be given as INADDR_ANY to mean "this host." The distinguished address INADDR_BROADCAST is allowed as a shorthand for the broadcast address on the primary network if the first network configured supports broadcast.

The Internet protocol family is comprised of the IP transport protocol, Internet Control Message Protocol (ICMP), Transmission Control Protocol (TCP), and User Datagram Protocol (UDP). TCP is used to support the *sock_stream* abstraction while UDP is used to support the *sock_dgram* abstraction. IP and ICMP are not currently supported.

The 32-bit Internet address contains both network and host parts. It is frequency-encoded; the most-significant bit is clear in Class A addresses, in which the high-order 8 bits are the network number. Class B addresses use the high-order 16 bits as the network field, and Class C addresses have a 24-bit network part. Sites with a cluster of local networks and a connection to the Defense Advanced Research Project Agency (DARPA) Internet may choose to use a single network number for the cluster; this is done by using subnet addressing. The local (host) portion of the address is further subdivided into subnet and host parts. Within a subnet, each subnet appears to be an individual network; externally, the entire cluster appears to be a single, uniform network requiring only a single routing entry.

SEE ALSO

intro(7B), tcp(7B), udp(7B).
socket(2B) in the *CLIX Programmer's & User's Reference Manual.*
"Introductory Socket Tutorial", "An Advanced Socket Tutorial" in the *CLIX System Guide.*

CAVEATS

The Internet protocol support is subject to change as the Internet protocols develop. Users should not depend on details of the current implementation, but rather the services exported.

NAME

mailaddr - mail addressing description

DESCRIPTION

Mail addresses used by *sendmail*(1M) are based on the protocol described in the RFC822 document. These addresses are in the following general format:

user@domain

In the above, *domain* is a hierarchical dot-separated list of subdomains. Consider the following example:

jim@sys1.xyzcom.COM

This is interpreted from right to left: the message should go to the "COM" name tables and then to the "xyzcom" gateway, after which it should go to the local host "sys1". When the message reaches "sys1" it is delivered to the user "jim".

Abbreviation

Under certain circumstances the entire *domain* name may not be necessary. In general, anything following the first dot may be omitted if it is the same as the domain from which you are sending the message. For example, a user on **sys2.xyzcom.com** could send to **jim@sys1** without adding the **xyzcom.com** because both systems are in the same domain.

Compatibility

Certain old address formats are converted to this format to provide compatibility with older mail systems. In particular, *host:user* is converted to *user@host* to be consistent with the *rcp*(1) command.

Also, the syntax *host!user* is converted to *user@host*.**UUCP** This syntax is normally converted back to the *host!user* form before being sent to UUCP hosts.

Case Distinctions

Domain names (anything after the @ sign) may be given in any mixture of upper and lowercase with the exception of UUCP hostnames.

Route-addrs

Under some circumstances it may be necessary to route a message through several hosts to get it to the final destination. Normally this routing occurs automatically, but sometimes it is desirable to route the message manually. Addresses that show these relays are called *route-addrs*. These addresses use the following syntax:

@hosta,@hostb:user@hostc

This specifies that the message should be sent to "hosta", from there to "hostb", and finally to "hostc". This path is forced even if there is a more efficient path to "hostc".

Route-addrs occur frequently on return addresses, because these are generally added by the software at each host. It is possible to ignore all but the *user@domain* part of the address to determine the actual sender.

Postmaster

Every site is required to have a user or user alias designated as *postmaster* to which problems with the mail system may be addressed.

Other Networks

Some other networks can be reached by giving the name of the network as the last component of the domain. This is not a standard feature and may not be supported at all sites. For example, messages to CSNET or BITNET sites can often be sent to *user@host*.**CSNET** or *user@host*.**BITNET**, respectively.

SEE ALSO

sendmail(1M).
rcp(1) in the *CLIX Programmer's & User's Reference Manual.*
mail(1), mailx(1) in the *UNIX System V User's Reference Manual.*

CAVEATS

The RFC822 group syntax (*group:user1,user2,user3;*) is not supported except in the special case of *group*: because of a conflict with old Berknet-style addresses.

NAME

ns - Xerox Network Systems protocol family

DESCRIPTION

The *ns* protocol family is a collection of protocols layered on top of the Internet Datagram Protocol (IDP) transport layer using the Xerox Network Systems (XNS) address formats. The *ns* family provides protocol support for the *sock_stream* and *sock_dgram* socket types.

Addressing

ns addresses are 12-byte quantities, consisting of a 4-byte network number, a 6-byte host number, and a 2-byte port number. All numbers are stored in network-standard format. On the CLIPPER, these numbers are word and byte reversed. The include file **<netns/ns.h>** defines the *ns* address as a structure containing unions (for quicker comparisons).

Sockets in the *ns* protocol family use the following addressing structure:

```
struct sockaddr_ns {
        u_short           sns_family;
        struct ns_addr    sns_addr;
        char              sns_zero[2];
};
```

An *ns_addr* is composed as follows:

```
union ns_host {
        u_char            c_host[6];
        u_short           s_host[3];
};

union ns_net {
        u_char            c_net[4];
        u_short           s_net[2];
};

struct ns_addr {
        union ns_net      x_net;
        union ns_host     x_host;
        u_short           x_port;
};
```

Sockets may be created with an address of all zeroes to effect "wildcard" matching on incoming messages.

Protocols

The *ns* protocol family supported by the operating system is Sequenced Packet Protocol (SPP). SPP is used to support the *sock_stream* abstraction.

SEE ALSO

intro(7B), spp(7B).
intro(3B), byteorder(3B), getnetent(3B), getservent(3B) getprotoent(3B), gethostbyname(3B) in the *CLIX Programmer's & User's Reference Manual.*

Internet Transport Protocols, Xerox Corporation document XSIS-028112.
"Advanced Socket Tutorial" in the *CLIX System Guide*.

NAME

spp - Xerox Sequenced Packet protocol

SYNOPSIS

#include <sys/socket.h>
#include <netns/ns.h>

socket (AF_NS, SOCK_STREAM, 0);

DESCRIPTION

The Sequenced Packet Protocol (SPP) provides reliable, flow-controlled, two-way transmission of data. It is a byte-stream protocol used to support the *sock_stream* abstraction. SPP uses the standard *ns*(7B) address formats.

Sockets using the SPP protocol are either active or passive. Active sockets initiate connections to passive sockets. By default, SPP sockets are created active; to create a passive socket, the user must *listen*(2B) after binding the socket with the *bind*(2B) system call. Only passive sockets may use the *accept*(2B) call to accept incoming connections. Only active sockets may use the *connect*(2B) call to initiate connections.

Passive sockets may underspecify their location to match incoming connection requests from multiple networks. This technique, termed wildcard addressing, allows a single server to service clients on multiple networks. To create a socket that listens on all networks, the user must *bind*(2B) an *ns*(7B) address of all zeroes. The SPP port may still be specified at this time; if the port is not specified, the system will assign one. Once a connection has been established, the socket's address is fixed by the peer entity's location. The address assigned to the socket is the address associated with the network interface through which packets are being transmitted and received. Normally, this address corresponds to the peer entity's network.

Packets received with the attention bit sent are interpreted as out of band data. Data sent with **send(..., ..., ..., MSG_OOB)** causes the attention bit to be set.

SEE ALSO

ns(7B), intro(7B).
Section (2B) in the *CLIX Programmer's & User's Reference Manual.*

NAME

tcp - Internet Transmission Control protocol

SYNOPSIS

#include <sys/socket.h>
#include <netinet/in.h>

socket (AF_INET, SOCK_STREAM, 0);

DESCRIPTION

The Transmission Control Protocol (TCP) provides reliable, flow-controlled, two-way transmission of data. It is a byte-stream protocol used to support the *sock_stream* abstraction. TCP uses the standard Internet address format and provides a per-host collection of "port addresses". Thus, each address is composed of an Internet address specifying the host and network, with a specific TCP port on the host identifying the peer entity.

Sockets using the TCP protocol are either active or passive. Active sockets initiate connections to passive sockets. By default TCP sockets are created active; to create a passive socket the *listen*(2B) system call must be used after binding the socket with the *bind*(2B) system call. Only passive sockets may use the *accept*(2B) call to accept incoming connections. Only active sockets may use the *connect*(2B) call to initiate connections.

Passive sockets may underspecify their location to match incoming connection requests from multiple networks. This technique, termed wildcard addressing, allows a single server to provide service to clients on multiple networks. To create a socket that listens on all networks, the user must *bind*(2B) the Internet address INADDR_ANY. The TCP port may still be specified at this time; if the port is not specified, the system will assign one. Once a connection has been established, the socket's address is fixed by the peer entity's location. The address assigned the socket is the address associated with the network interface through which packets are being transmitted and received. Normally, this address corresponds to the peer entity's network.

SEE ALSO

intro(7B), inet(7B), ip(7B).
Section (2B) in the *CLIX Programmer's & User's Reference Manual.*

NAME

udp - Internet User Datagram protocol

SYNOPSIS

#include <sys/socket.h>
#include <netinet/in.h>

socket (AF_INET, SOCK_DGRAM, 0);

DESCRIPTION

The User Datagram Protocol (UDP) is a simple, unreliable datagram protocol used to support the *sock_dgram* abstraction for the Internet protocol family. UDP sockets are connectionless and are normally used with the *sendto*(2B) and *recvfrom*(2B) calls, though the *connect*(2B) call may also be used to fix the destination for future packets. In which case, the *recv*(2B), *read*(2B), or *readv*(2B) and *send*(2B), *write*(2), or *writev*(2B) system calls may be used.

UDP uses the standard Internet address format and provides a per-host collection of port addresses. Note that the UDP port space is separate from the *tcp*(7B) port space. A UDP port may not be "connected" to a *tcp*(7B) port. In addition, broadcast packets may be sent (assuming the underlying network supports this) by using a reserved broadcast address; this address is network-interface-dependent.

SEE ALSO

inet(7B), intro(7B).
Section (2B) in the *CLIX Programmer's & User's Reference Manual.*

7A

NAME

intro - introduction to asynchronous interfaces

DESCRIPTION

This section describes the asynchronous interfaces. These interfaces support device input, output, and control. Providing an asynchronous interface allows a process to have many operations active at once, thus increasing overall performance.

SEE ALSO

intro(3A) in the *CLIX Programmer's & User's Reference Manual.*

NOTES

Since these interfaces are not standard device drivers, no device files are associated with the hardware they control. Refer to the specific asynchronous interface descriptions for information concerning hardware accesses.

NAME

xaux - asynchronous serial interface driver

DESCRIPTION

xaux provides an asynchronous interface for reading and writing raw unprocessed data from the system serial ports.

A system may support three or four serial ports, depending on the platform and system configuration.

SEE ALSO

aux(7S).
aux_open(3A), aux_read(3A), aux_write(3A), aux_cancel_modem(3A), aux_cancel(3A), aux_break(3A), aux_rawrd(3A), aux_modem(3A), aux_close(3A) in the *CLIX Programmer's & User's Reference Manual*.

CAVEATS

Serial port characteristics cannot be established through this interface. The *ioctl*(2) interface can be used to establish port characteristics on the *aux*(7S) special file. Note that the xaux_open(3A) must occur before the *open*(2) of the associated *aux*(7S) serial device.

NAME

xcnv - convolution filter

DESCRIPTION

The Convolution Filter (CNV) performs a convolution operation on the data flowing through the raster processing pipeline. The CNV board in the lowest Shared Resource (SR) Bus slot number supports channel 0, the next CNV board supports channel 1, and so on.

xcnv allows a process to directly access the control registers of a specified CNV board. All convolution parameters are established through the mapped registers.

SEE ALSO

cnv_close(3A), cnv_open(3A) in the *CLIX Programmer's & User's Reference Manual.*

NAME

xcsi - control status interface

DESCRIPTION

xcsi is a DR11-based communication interface that uses the Control Status Interface (CSI) port that resides on the Image System Interface (ISI) board. The ISI board may reside on the Shared Resource (SR) bus of Intergraph series 300 and 400 workstations and servers. *xcsi* is used for high-speed transfer of command and status packets between the workstation and various special-purpose hardware.

The hardware is a subset of the standard DR11 interface. *xcsi* defines a protocol that uses the DR11 function and status lines along with the attention interrupts. The workstation is a master and can only support connections to slave DR11 devices. In addition, communication depends on a standard header common to all command and status packets. The header indicates the source and destination of each packet, along with packet size and validity checking information. The definition of this header is in **<sys/xio/xcsi.h>** and its fields are shown below:

```
short   h_cmd;        /* CSI command/status ID */
short   h_mswcnt;     /* MSW of command size */
short   h_lswcnt;     /* LSW of command size */
short   h_src;        /* source ID */
short   h_dest;       /* route code */
short   h_magic0;     /* 0x5555 sanity check */
short   h_magic1;     /* 0xaaaa sanity check */
short   h_seq0;       /* relative start time */
short   h_seq1;       /* relative finish time */
short   h_checksum;   /* header checksum sanity check */
```

Each ISI board supports a single CSI port. The CSI port on the ISI board in the lowest SR slot number is referred to as channel 0. The next ISI board supports channel 1 and so forth. Each *xcsi* channel may be opened and used by many processes at a time.

To send commands to the CSI port, use *csi_cmd*(3A) specifying a currently open channel from *csi_open*(3A). Incoming status packets are matched with outgoing commands based on command ID and source/route codes. Refer to *csi_dstat*(3A) for commands that may respond with multiple status packets. If the slave device needs special attention, it may send an unsolicited status packet. Refer to *csi_ustat*(3A) for information on intercepting these packets. Refer to *csi_death*(3A) to monitor hardware problems at run time.

SEE ALSO

csi_cancel(3A), csi_ccan(3A), csi_close(3A), csi_cmd(3A), csi_death(3A), csi_dstat(3A), csi_open(3A), csi_reset(3A), csi_status(3A), csi_ucan(3A), csi_ustat(3A) in the *CLIX Programmer's & User's Reference Manual.*

NAME

xfpe - FPE coprocessor interface

DESCRIPTION

xfpe is an asynchronous interface driver that provides access to a Floating-Point Engine (FPE) coprocessor. The FPE coprocessor is an optional processor board that performs high-precision numeric operations for specific applications.

xfpe provides a channel scheme to allow a process to access an FPE. Using *fpe_coproc_alloc*(3A), a process obtains a channel number to use for all subsequent operations. This channel number enables the driver to associate a process's request with a specific coprocessor.

xfpe provides a mechanism for a process to load application-specific microcode images into an FPE executable memory space. The driver supports up to NDID—1 loaded images if enough free executable memory space is on the FPE. The available space depends on the size of each microcode image.

Data may be transferred to the FPE by either direct writes to the first-in first-out (FIFO) register located on the FPE board, or by a call to CLIX to transfer the data directly from user memory to the FPE FIFO.

SEE ALSO

fpe_coproc_alloc(3A), fpe_coproc_dealloc(3A), fpe_cancel_dma(3A), fpe_did_load(3A), fpe_did_unload(3A), fpe_write_dma(3A) in the *CLIX Programmer's & User's Reference Manual.*

NAME

xgpib - GPIB driver

DESCRIPTION

xgpib is an asynchronous interface driver that provides bus control, device control, and data transfer functions for the IEEE 488-1978 standard digital interface bus. General Purpose Interface Bus (GPIB) is the common term for the IEEE 488 interface.

xgpib supports a maximum of NGPIB interface buses; NGPIB normally equals one. Using a channel approach, the driver associates the GPIB board in the lowest-numbered hardware slot with channel zero, the next GPIB board corresponds to channel one, and so forth. Generally, the channel number specified when using the GPIB library functions is zero, since only one GPIB board is typically in a system. A process allocates a channel using the *gpib_open*(3A) function.

xgpib accepts GPIB device addresses and performs the IEEE 488 protocol based on the function. For example, a process to clear a device on the bus simply sends the device's address to the driver using the *gpib_clear*(3A) function. The driver executes the protocol by sending the required bus commands. This feature spares the user the task of formulating the protocols for every bus operation.

xgpib supports device-specific data transfers using hardware implemented Direct Memory Access (DMA) transfers. The DMA capability provides expedient data throughput on the bus and allows the system CPU to perform other operations.

xgpib also provides a timeout capability with some functions allowing a process to implement error recovery procedures. The timeout value specifies the number of 1/60-second intervals *xgpib* waits before aborting the function. A timeout value of zero always disables the timeout feature.

SEE ALSO

gpib_cancel(3A), gpib_clear(3A), gpib_close(3A), gpib_cmd(3A), gpib_local(3A), gpib_lockout(3A), gpib_open(3A), gpib_ppconf(3A), gpib_ppreq(3A), gpib_ppuconf(3A), gpib_read(3A), gpib_remote(3A), gpib_reset(3A), gpib_service(3A), gpib_spreq(3A), gpib_trigger(3A), gpib_write(3A) in the *CLIX Programmer's & User's Reference Manual.*

WARNINGS

xgpib requires primary device addresses and assumes the system is the only GPIB controller.

The data transfer functions that use the DMA capability (*gpib_read*(3A) and *gpib_write*(3A)) have specific restrictions. Data buffers must begin on long-word (4-byte multiple) boundaries and read buffers must end on long-word boundaries.

NAME

xnlf - non-linear filter

DESCRIPTION

The Non-Linear Filter (NLF) performs a nonlinear filter operation on the data flowing through the raster processing pipeline. The NLF board in the lowest Shared Resource (SR) Bus slot number supports channel 0, the next NLF board supports channel 1, and so on.

xnlf allows a process to directly access the control registers for a specified NLF board. All filter parameters are established using the mapped registers following a successful *nlfopen*(3A).

SEE ALSO

nlf_close(3A), nlf_open(3A) in the *CLIX Programmer's & User's Reference Manual.*

NAME

xpdi - processed data interface

DESCRIPTION

The Processed Data Interface (PDI) is a bidirectional, half-duplex, 8-bit parallel port that physically resides on the Image System Interface (ISI) board. The ISI board may reside on the Shared Resource (SR) bus of Intergraph series 300 and 400 workstations and servers. *xpdi* uses the PDI port to transfer large amounts of data to or from external devices such as plotters, scanners, and various specialty hardware.

Each ISI board supports a single PDI port. The PDI port on the ISI board in the lowest SR slot number is referred to as channel 0, the next ISI board supports channel 1, and so on.

To write to the PDI port, use *pdi_write*(3A), specifying a currently-open channel from *pdi_open*(3A).

To read from the PDI port, use *pdi_read*(3A), specifying a currently-open channel from *pdi_open*(3A). The PDI read interface is implemented with a hardware capability to reject large amounts of data. For example, if a 1/4-inch strip is needed from a scanner that always sends inch-wide strips, the PDI hardware can be set up to transfer only the data pertaining to the first 1/4-inch of each line in the strip. The remainder of each line is discarded. This reduces memory and time requirements. Note that this rejection is a function of the resolution and linewidth specified with *pdi_setup*(3A).

xpdi also supports the transfer of data from an external device directly to a graphics window on an Integrated Frame Buffer (IFB). Refer to *pdi_ifb*(3A) for more information on this capability.

The PDI port generates an output signal that may be switched by the process controlling it. One type of plotter uses this signal to envelope all data transfers pertaining to a single plot. This signal is referred to as the write_valid signal and is manipulated with *pdi_setup*(3A).

SEE ALSO

pdi_close(3A), pdi_ifb(3A), pdi_open(3A), pdi_read(3A), pdi_write(3A), pdi_cancel(3A), pdi_setup(3A) in the *CLIX Programmer's & User's Reference Manual.*

NAME

xplot - asynchronous plotting interface

DESCRIPTION

xplot provides a mechanism to write data or control information to the parallel port. The interface supports Versatec, Centronics, and Intergraph Differential.

The hardware to support the parallel port resides either on the Raster Operations Processor (ROP) graphics board, or on a dedicated plotter board. The plotter board will only be available if a ROP board is not present.

SEE ALSO

plot_ctrl_rop(3A), plot_data_rop(3A), plot_ctrl(3A), plot_data(3A) in the *CLIX Programmer's & User's Reference Manual.*

NAME

xrle - run length encoding interface

DESCRIPTION

The Run Length Encoding (RLE) board provides a hardware-driven mechanism to run length encode data with various formats from the raster processing pipeline to virtual memory. The RLE board in the lowest Shared Resource (SR) Bus slot number supports channel 0, the next RLE board supports channel 1, and so on.

Parameters pertaining to line width, swath size, margins and data format must be specified with *rle_setup*(3A) before any data is processed.

To run length encode data from the pipe to memory, use *rle_pipe_mem*(3A) specifying a channel currently open from *rle_open*(3A).

SEE ALSO

rle_cancel(3A), rle_close(3A), rle_open(3A), rle_pipe_mem(3A), rle_setup(3A) in the *CLIX Programmer's & User's Reference Manual.*

NAME

xsif - scanner interface

DESCRIPTION

The Scanner Interface (SIF) provides an 8-bit parallel interface to a scanner as well as a memory DMA interface and a 32-bit FIFO for pipelined raster processing capability. The scanner may be accessed in full RGB color mode, or each color component may be accessed individually.

xsif uses the SIF board to transfer large amounts of data from memory to a raster processing pipeline, a scanner to memory, or a scanner to a raster processing pipeline.

The SIF board in the lowest Shared Resource (SR) Bus slot number supports channel 0, the next SIF board supports channel 1, and so on.

To transfer data from memory to the pipe, use *sif_mem_pipe*(3A), specifying a channel currently open from *sif_open*(3A).

To transfer data from the scanner to memory, use *sif_scan_mem*(3A), specifying a channel currently open from *sif_open*(3A).

To transfer data from the scanner to the pipe, use *sif_scan_pipe*(3A), specifying a channel currently open from *sif_open*(3A).

The SIF scanner interface is implemented with a hardware capability to reject large amounts of data. For example, if a 1/4-inch strip is needed from a scanner that always sends inch-wide strips, the SIF hardware can be set up to transfer only the data pertaining to the first 1/4 inch of each line in the strip. The remainder of each line is discarded. This reduces memory and time requirements for the workstation. This rejection is a function of the resolution and linewidth specified with *sif_setup*(3A). *sif_setup*(3A) also controls swath size and color/mono modes.

SEE ALSO

sif_cancel(3A), sif_scan_mem(3A), sif_mem_pipe(3A), sif_close(3A), sif_scan_pipe(3A), sif_open(3A), sif_setup(3A) in the *CLIX Programmer's & User's Reference Manual.*