UNICOS® File Formats and Special Files Reference Manual

SR–2014 10.0

Portions of this product may still be in development. The existence of those portions still in development is not a commitment of actual release or support by Cray Research, Inc. Cray Research, Inc. assumes no liability for any damages resulting from attempts to use any functionality or documentation not officially released and supported. If it is released, the final form and the time of official release and start of support is at the discretion of Cray Research, Inc.

The UNICOS operating system is derived from UNIX® System V. The UNICOS operating system is also based in part on the Fourth Berkeley Software Distribution (BSD) under license from The Regents of the University of California.

New Features

UNICOS® File Formats and Special Files Reference Manual SR–2014 10.0

This rewrite of the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014, supports the 10.0 release of the UNICOS operating system. The following changes have been made:

- Addition of `arrayd.conf`(5) to support the array session feature.
- Revision of `quota`(5) to support the optional aggregate quota feature.
- Addition of `text_tapeconfig`(5) to replace `tapeconfig`. The default configuration file is now `text_tapeconfig`. This updated command provides information about overcommitted mount requests: the new `-a` option outputs device status. The `DEVICE` statement provides support for IBM ESCON tape devices: the new `timeout` parameter specifies the time-out value in seconds that the ESCON IOP waits for a response from the channel, and the `type` parameter now accepts 3590 as a type.

Record of Revision

Version	*Description*
1.0	March 1986 Original Printing. This documentation supports the UNICOS 1.0 release.
1.1	June 1986 Online documentation only. This documentation supports the UNICOS 1.1 release.
2.0	October 1986 This documentation supports the UNICOS 2.0 release.
3.0	July 1987 This documentation supports the UNICOS 3.0 release.
4.0	July 1988 This documentation supports the UNICOS 4.0 release.
5.0	February 1989 This documentation supports the UNICOS 5.0 release.
6.0	January 1991 This documentation supports the UNICOS 6.0 release.
7.0	July 1992 This documentation supports the UNICOS 7.0 release.
8.0	January 1994 This documentation supports the UNICOS 8.0 release.
9.0	September 1995 This documentation supports the UNICOS 9.0 release.
10.0	November 1997 This documentation supports the UNICOS 10.0 release.

Preface

This publication documents UNICOS release 10.0 running on CRAY PVP and CRAY T3D systems.

Related publications

The following man page manuals contain additional information that may be helpful.

> **Note:** For the UNICOS 10.0 release, man page reference manuals are not orderable in printed book form. Instead, they are available as printable PostScript files provided on the same DynaWeb CD as the rest of the supporting documents for this release. Individual man pages are still available online and can be accessed by using the `man`(1) command.

- *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011
- *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012
- *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022
- *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

The following ready references are available in printed form from the Distribution Center:

- *UNICOS User Commands Ready Reference*, Cray Research publication SQ–2056
- *UNICOS System Libraries Ready Reference*, Cray Research publication SQ–2147
- *UNICOS System Calls Ready Reference*, Cray Research publication SQ–2215
- *UNICOS Administrator Commands Ready Reference*, Cray Research publication SQ–2413

Ordering Cray Research publications

The *User Publications Catalog*, Cray Research publication CP–0099, describes the availability and content of all Cray Research hardware and software documents that are available to customers. Cray Research customers who subscribe to the Cray Inform (CRInform) program can access this information on the CRInform system.

To order a document, either call the Distribution Center in Mendota Heights, Minnesota, at +1–612–683–5907, or send a facsimile of your request to fax number +1–612–452–0141. Cray Research employees may send electronic mail to `orderdsk` (UNIX system users).

Customers who subscribe to the CRInform program can order software release packages electronically by using the `Order Cray Software` option.

Customers outside of the United States and Canada should contact their local service organization for ordering and documentation information.

Conventions

The following conventions are used throughout this document:

Convention	Meaning
`command`	This fixed-space font denotes literal items such as commands, files, routines, path names, signals, messages, and programming language structures.
`manpage`(*x*)	Man page section identifiers appear in parentheses after man page names. The following list describes the identifiers:

1	User commands
1B	User commands ported from BSD
2	System calls
3	Library routines, macros, and opdefs
4	Devices (special files)
4P	Protocols
5	File formats
7	Miscellaneous topics
7D	DWB-related information

	8 Administrator commands
	Some internal routines (for example, the `_assign_asgcmd_info`() routine) do not have man pages associated with them.
variable	Italic typeface denotes variable entries and words or concepts being defined.
`user input`	This bold, fixed-space font denotes literal items that the user enters in interactive sessions. Output is shown in nonbold, fixed-space font.
[ ]	Brackets enclose optional portions of a command or directive line.
...	Ellipses indicate that a preceding element can be repeated.

The following machine naming conventions may be used throughout this document:

Term	Definition
Cray PVP systems	All configurations of Cray parallel vector processing (PVP) systems.
Cray MPP systems	All configurations of the CRAY T3D series. The UNICOS operating system is not supported on CRAY T3E systems. CRAY T3E systems run the UNICOS/mk operating system.
All Cray Research systems	All configurations of Cray PVP and Cray MPP systems that support this release.

The default shell in the UNICOS and UNICOS/mk operating systems, referred to in Cray Research documentation as the *standard shell*, is a version of the Korn shell that conforms to the following standards:

- Institute of Electrical and Electronics Engineers (IEEE) Portable Operating System Interface (POSIX) Standard 1003.2–1992
- X/Open Portability Guide, Issue 4 (XPG4)

The UNICOS and UNICOS/mk operating systems also support the optional use of the C shell.

Cray UNICOS Version 10.0 is an X/Open Base 95 branded product.

Man page sections

The entries in this document are based on a common format. The following list shows the order of sections in an entry and describes each section. Most entries contain only a subset of these sections.

Section heading	Description
NAME	Specifies the name of the entry and briefly states its function.
SYNOPSIS	Presents the syntax of the entry.
IMPLEMENTATION	Identifies the Cray Research systems to which the entry applies.
STANDARDS	Provides information about the portability of a utility or routine.
DESCRIPTION	Discusses the entry in detail.
NOTES	Presents items of particular importance.
CAUTIONS	Describes actions that can destroy data or produce undesired results.
WARNINGS	Describes actions that can harm people, equipment, or system software.
ENVIRONMENT VARIABLES	Describes predefined shell variables that determine some characteristics of the shell or that affect the behavior of some programs, commands, or utilities.
RETURN VALUES	Describes possible return values that indicate a library or system call executed successfully, or identifies the error condition under which it failed.
EXIT STATUS	Describes possible exit status values that indicate whether the command or utility executed successfully.
MESSAGES	Describes informational, diagnostic, and error messages that may appear. Self-explanatory messages are not listed.
ERRORS	Documents error codes. Applies only to system calls.

FORTRAN EXTENSIONS	Describes how to call a system call from Fortran. Applies only to system calls.
BUGS	Indicates known bugs and deficiencies.
EXAMPLES	Shows examples of usage.
FILES	Lists files that are either part of the entry or are related to it.
SEE ALSO	Lists entries and publications that contain related information.

Reader comments

If you have comments about the technical accuracy, content, or organization of this document, please tell us. You can contact us in any of the following ways:

- Send us electronic mail at the following address:

  `publications@cray.com`

- Contact your customer service representative and ask that an SPR or PV be filed. If filing an SPR, use `PUBLICATIONS` for the group name, `PUBS` for the command, and `NO-LICENSE` for the release name.

- Call our Software Publications Group in Eagan, Minnesota, through the Customer Service Call Center, using either of the following numbers:

  1–800–950–2729 (toll free from the United States and Canada)

  +1–612–683–5600

- Send a facsimile of your comments to the attention of “Software Publications Group” in Eagan, Minnesota, at fax number +1–612–683–5599.

We value your comments and will respond to them promptly.

CONTENTS

NAME

`intro` – Introduction to special files

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The entries in this section describe the characteristics of the device interfaces (device drivers) and corresponding hardware devices or pseudo devices in UNICOS; there is one entry or set of related entries per page.

UNICOS devices and pseudo devices are represented by special files in the `/dev` directory. With a few exceptions, each hardware device is represented by one or more files in `/dev`. Examples of devices are disk drives, which are represented by the special files in the `/dev/dsk` directory and are described by the `dsk`(4) entry. *Pseudo devices* are device drivers that have no associated hardware but which behave in much the same way as a hardware device. Examples of pseudo devices are the null pseudo device, `/dev/null` (described by the `null`(4) entry), and the pseudo terminals, located in the `/dev/pty` directory (described by the `pty`(4) entry).

Three types of special files exist: block special files, character special files, and FIFO special files (named pipes). This manual does not discuss FIFO special files; for information on these files, see `pipe`(2).

On *block devices*, data read from or written to the device is moved through a cache of system buffers. In contrast, devices that do not use the system buffer cache are *character devices*. (Character devices do not necessarily move data 1 character at a time; in fact, very large blocks (track-sized or cylinder-sized) are often used.) The unbuffered I/O of character devices is often called *raw* I/O mode.

On CRAY Y-MP systems, disks and RAM disks are the only block devices supported. These devices are documented in `dsk`(4) and `ram`(4). The supported block devices are disks, buffer memory resident (BMR) file systems, the SSD solid-state storage device, and RAM disks. The `dsk`(4) and `ram`(4) entries describe these devices.

You can use disk drives as character devices instead of block devices; in this case, the system buffer cache is not used, and the data is moved directly between the device and the user's buffer. The `fcntl`(2) system call is available to open the block special file in raw mode (see `fcntl`(5) and `dsk`(4)).

Most devices and pseudo devices can do read and write operations; many can do additional operations. The capabilities of each device are discussed in the entry for that device.

Many devices allow further manipulation of the device through the special file with the `ioctl`(2) system call. For example, a process can issue an `ioctl` request to return the status of a device; the `ioctl` request `CPUSTAT` returns the status of the target CPU (see `cpu`(4)). The `ioctl` requests are device-dependent, and they are discussed in the entry for each device if appropriate.

Special files are created using the `mknod`(8) command, which builds a directory entry and an inode for a device. For specific information on creating devices, see the `mknod`(8) command and the system installation bulletin for your UNICOS release.

SEE ALSO

`cpu`(4), `fcntl`(2), `ioctl`(2), `pipe`(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

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

NAME

cpu – Interface to special CPU functions

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The cpu driver allows control of many different aspects of the CPU environment in which processes run. Some of the functions refer to a specific physical CPU. The naming convention for the special files corresponding to CPUs is as follows:

CPU 0	/dev/cpu/0
CPU 1	/dev/cpu/1
CPU 2	/dev/cpu/2
CPU 3	/dev/cpu/3
CPU 4	/dev/cpu/4
CPU 5	/dev/cpu/5
CPU 6	/dev/cpu/*ncpus*

The following usage, while still accepted, may not be supported in future releases.

CPU 0	/dev/cpu/a	CPU 8	/dev/cpu/i
CPU 1	/dev/cpu/b	CPU 9	/dev/cpu/j
CPU 2	/dev/cpu/c	CPU 10	/dev/cpu/k
CPU 3	/dev/cpu/d	CPU 11	/dev/cpu/l
CPU 4	/dev/cpu/e	CPU 12	/dev/cpu/m
CPU 5	/dev/cpu/f	CPU 13	/dev/cpu/n
CPU 6	/dev/cpu/g	CPU 14	/dev/cpu/o
CPU 7	/dev/cpu/h	CPU 15	/dev/cpu/p

Some of the functions refer to a CPU in which the specified process (or processes) may happen to execute. For those functions, you should use the special file /dev/cpu/any. This special file provides a generic interface to any CPU.

For sites with more than one CPU, create as many CPU devices as needed, changing the name and minor device number in sequence.

The only valid system calls to the cpu driver are open(2), close(2), and ioctl(2).

The available ioctl requests are defined in the sys/cpu.h include file.

For the following requests, *arg* is interpreted as a pointer to an exchange packet. The following ioctl requests are supported for a specific CPU:

CPU_DOWN Disables the CPU.

CPU_START
: Copies the exchange package to which *arg* points to the system diagnostic and stores it. The exchange information is assumed to be absolute address 0 of the stand-alone program. The target CPU must be in a down state. The base address (BA) in the exchange package is considered as relative to the caller's base address; the limit address (LA) is set to that of the caller.

  CPU_START is disabled for CRAY T90 series architecture. This ioctl request will return a value of EINVAL.

CPU_STAT
: Copies the system's user exchange package to *arg*. CPU_STAT copies the user exchange information only if the target CPU exits.

CPU_DSTAT
: Copies the system's diagnostic exchange package to *arg*. CPU_DSTAT copies the diagnostic exchange information only if the target CPU exits.

CPU_STOP
: Halts the target CPU; *arg* is not used but must be supplied. The CPU must have been started by using PU_START.

  CPU_STOP is disabled for CRAY T90 series architecture. This ioctl request will return a value of EINVAL.

CPU_UP
: Reenables the CPU. Permits the CPU to be scheduled for normal processing. The CPU must have been downed previously (CPU_DOWN).

CPU_UP_S_CACHE
: Enables scalar cache for the specified CPU. (CRAY T90 and CRAY J90 series)

CPU_DN_S_CACHE
: Disables scalar cache for the specified CPU. (CRAY T90 and CRAY J90 series)

The following ioctl requests apply only to CRAY Y-MP systems that are supported on the generic device (/dev/cpu/any). These requests require the following structure (except for the CPU_CLRTMR request). This structure is defined in the sys/cpu.h include file, as follows:

```
struct  cpudev  {
        int   cat;         /* category   */
        int   id;          /* identifier */
        long  word;        /* parameter  */
        long  cpu_word1;   /* parameter  */
        long  cpu_word2;   /* parameter  */
        long  cpu_word3;   /* parameter  */
        long  cpu_word4;   /* parameter  */
};
```

CPU_CLRRT
: Clears real-time status from the specified process or process group.

CPU_CLRTMR
: Removes the calling process from the interval timer queue. The cpudev structure is not required.

`CPU_CLUSTER`	Selects the clusters specified by the bit mask in `word` for the ID and category specified by `id` and `cat`. If `id` is 0, the current process ID is assumed. Bit 2^k in the mask refers to cluster *k*. The resulting bit mask is returned. Clusters 0 and 1 cannot be selected.
`CPU_DEDICATE`	Dedicates the CPUs specified by the bit mask in `word` to the ID and category specified by `id`. If `id` is 0, the current process ID is assumed. Bit 2^k in the mask refers to CPU *k*. If the mask contains 0 bits for CPUs that are already dedicated, those CPUs are released from dedication. The resulting bit mask is the return value of `CPU_DEDICATE`.
`CPU_GETMODE`	Returns a bit mask of the current mode settings for a process in a process group. The bit positions in the mask (right-justified) are as follows:

`UXP_MON`	0	Monitor mode
`UXP_BDM`	1	Bidirectional memory
`UXP_EMA`	2	Extended mode addressing
`UXP_AVL`	3	Second vector logical
`UXP_IFP`	4	Interrupt on floating-point error
`UXP_IOR`	5	Interrupt on operand range error
`UXP_ICM`	6	Interrupt on correctable memory errors
`UXP_IUM`	7	Interrupt on uncorrectable memory errors
`UXP_IMM`	8	Interrupt monitor mode
`UXP_RPE`	9	Register parity interrupts enabled
`UXP_SCE`	10	Scalar cache enabled
`UXP_IIO`	11	I/O interrupts enabled
`UXP_IPC`	12	Programable clock interrupts enabled
`UXP_IAM`	13	AMI interrupts enabled
`UXP_IXI`	14	Exceptional input interrupt enabled (IEEE)
`UXP_INX`	15	Inexact interrupt enabled (IEEE)
`UXP_IUN`	16	Underflow interrupt enabled (IEEE)
`UXP_IOV`	17	Overflow interrupt enabled (IEEE)
`UXP_IDV`	18	Divide by zero interrupt enabled (IEEE)
`UXP_INV`	19	Invalid interrupt enabled (IEEE)
`UXP_RM0`	20	Round mode 0 set (IEEE)
`UXP_RM1`	21	Round mode 1 set (IEEE)

`CPU_QDOWN`	Returns a mask of the CPUs down. The number of CPUs configured is returned in `word`. The mask of down CPUs is returned in `cpu_word1`.

`CPU_RTFRAME` Enables least-time-to-go scheduling for the calling process. Requires the following parameters:

`cpuctl.rtframe`	Count of milliseconds in frame
`cpuctl.rtitm`	Count of milliseconds of input time
`cpuctl.rtctm`	Count of milliseconds of required computer time
`cpuctl.rtotm`	Count of milliseconds of required output time

A total frame time in `os_hz` units is passed in `cpuctl.rtframe`. This value is used for least-time-to-go scheduling of real-time processes. An optional signal number may be passed in `word1`. If nonzero, this signal will be sent each time the frame time expires. You can send this command at any time to resynchronize the frame start point. The category must be `C_PROC`, the ID must be 0 or the caller's process ID, and the process must be real time.

`CPU_RTPERMIT` Used by super-user processes to bestow permission to nonsuper-user processes; used by process groups to request real-time status.

`CPU_SELECT` Selects the CPUs specified by the bit mask in `word` for the ID and category specified by `id` and `cat`. If `id` is 0, the current process ID is assumed. Bit 2^k in the mask refers to CPU *k*. The resulting bit mask is the return value of `CPU_SELECT`.

`CPU_SETMODE` Sets mode bits. `word` is a bit mask. The value 1 in a bit position enables the mode; the value 0 disables the mode. The bit positions in the mask are the same as those in the `ioctl` request `CPU_GETMODE`. If the calling process is not a super-user process, `UXP_MON`, `UXP_IIO`, `UXP_IPC`, and `UXP_IAM` set to 1 returns the `EPERM` error.

`CPU_SETRT` Marks the ID and category specified by `id` and `cat` as being real-time processes. Real-time processes are scheduled from a separate run queue that is always checked before the default run queue. Priorities for real-time processes are initialized as are other processes, but they are not adjusted by the system. The run queue can be ordered by a change in nice values (see `nice`(2)). Real-time processes have permissions that allow them to use clock and CPU dedication even though they may not be a super user. The `EPERM` error is returned if the calling process is not a super-user process or does not have permission (see `CPU_RTPERMIT`).

`CPU_SETTMR` Places the calling process on the interval timer queue. The interval (in milliseconds) is in `word`. The process then receives `SIGALRM` signals at the specified interval.

`CPU_GETMAXERR`	Returns values from kernel `maxerrint` table. The maximum `PRE` count is returned in `cpu_word1`. The maximum `ORE` count is returned in `cpu_word2`. The maximum `ERR` count is returned in `cpu_word3`. The values in the `maxerrint` table specify the maximum count of Program Range Errors, Operand Range Errors, and Error Exits that are allowed for a process in one connection to a CPU. If any of the counts is exceeded, the process is sent a `SIGKILL` signal. A `maxerrint` table value of 0 means that no limit is enforced for that particular error.
`CPU_SETMAXERR`	Sets values into the kernel `maxerrint` table. `word` is a bit mask that indicates which fields should be set. If bit 2**0 of the mask is set, the maximum `PRE` count is set to the value in `cpu_word1`. If bit 2**1 of the mask is set, the maximum `ORE` count is set to the value in `cpu_word2`. If bit 2**2 of the mask is set, the maximum `ERR` count is set to the value in `cpu_word3`.

In addition to the standard `ioctl` error codes (see `ioctl`(2)), the following are errors that cause an `ioctl` request to fail:

`EFAULT`	Exchange information address (*arg*) is out of the user's memory area
`EINVAL`	Nonexistent CPU is requested, category is an unknown type, or timer interval is 0
`ENODEV`	Last CPU is being stopped by using `CPUSTOP`
`ENOENT`	Timer queue is full
`EPERM`	User is not super user or does not own the device for device-specific (CPU-specific) requests
`ESRCH`	No existing process matches the category and ID specified

FILES

`/dev/cpu/0`	Device driver for CPU 0
`/dev/cpu/1`	Device driver for CPU 1
`/dev/cpu/2`	Device driver for CPU 2
`/dev/cpu/3`	Device driver for CPU 3
`/dev/cpu/4`	Device driver for CPU 4
`/dev/cpu/5`	Device driver for CPU 5
.	
.	
.	
`/dev/cpu/32`	Device driver for CPU 32
`/dev/cpu/any`	Device driver for any CPU

`/usr/include/sys/category.h`	CPU `ioctl` request definitions
`/usr/include/sys/cpu.h`	

SEE ALSO

`close`(2), `ioctl`(2), `nice`(2), `open`(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

NAME

disk – Physical disk interface

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

The /dev/disk device is used as the interface to all of the real physical disk devices configured. The ioctl(2) system call is used to issue requests to individual devices by passing the ASCII name of the physical device in the control structure to the ddcntl.c device driver. The ASCII names of the physical devices are those configured in cf/conf.*SN*.c (*SN* is the mainframe serial number) or the I/O subsystem (IOS) parameter file.

The control structure used for the ioctl system call is defined in the sys/ddcntl.h include file, as follows:

```
struct ddctl {
        daddr_t    dc_bno;     /* Block number           */
        waddr_t    dc_buff;    /* Buffer to return data  */
        int        dc_off;     /* Offset                 */
        long       dc_count;   /* Count                  */
        long       dc_name;    /* Physical device name   */
        int        dc_size;    /* Size                   */
        int        dc_type;    /* Cache type             */
};

#include <sys/types.h>
#include <sys/fcntl.h>
#include <sys/ddcntl.h>

fdes = open(DISKDEV, O_RDWR);
ioctl(fdes, cmd, &ddcntl)
```

Following is a description of the available ioctl requests:

DC_ACACHE Adds a cache to a logical device dc_name (for logical device caching). dc_count is the number of cache buffers, dc_size is the size of each buffer, and dc_type is the type of cache (DDBMR or DDSSD).

DC_AFLW Adds the dc_bno block on the dc_name device to the flaw map.

DC_DFLW Removes the dc_bno block on the dc_name device from the flaw map.

DC_DOWN Configures the dc_name device to DOWN. This causes any I/O requests to that device to fail and return an error.

DC_GO	Restarts the dc_name device.
DC_MAINT	Places slice dc_count of the logical device specified in dc_off into or out of maintenance mode, depending on whether dc_type is nonzero or 0. You can use this command only on slices that reside on physical devices that are configured DOWN or RONLY. The call sets bit SI_OFLD in the si_flags word of the size structure associated with the slice (see the iobuf.h file). Bit SI_OFLD can be read by using the DC_MAP system call associated with the disk device. To clear the bit, either issue another DC_MAINT call or configure UP the physical device that contains the slice. The fsoffload(8) command uses the DC_MAINT call.
DC_RCACHE	Removes all cache from a logical device (for logical device caching).
DC_READ	Reads the number of bytes (which must be a sector multiple) specified in dc_count to the address specified in dc_buff from the dc_name device.
DC_RFLW	Replaces a flawed block with a new spares block.
DC_RIOBUF	Returns the iobuf structure, which is defined in the sys/iobuf.h include file, for the device name in dc_name into the user's buffer at dc_buff.
DC_RMAP	Returns the ldmap entry and slice entry for the logical device specified by the minor device number in dc_off to the buffer in dc_buff.
DC_RONLY	Sets device dc_name so that new space will not be allocated on that device.
DC_RTAB	Returns to the buffer in dc_buff the iobuf structures for all configured physical devices.
DC_SSDOFF	Configures the SSD channel specified in dc_off to DOWN.
DC_SSDON	Configures the SSD channel specified in dc_off to UP.
DC_SSDTAB	Returns the SSD control table (ssdconf) to the buffer in dc_buff.
DC_SSTHRSH	Sets the SSD threshold between synchronous and asynchronous transfers to dc_count sectors.
DC_STOP	Stops the dc_name device. This causes all I/O requests to that device to be queued until the device is configured up through DC_UP.
DC_SYNCALL	Flushes all logical device caches to disk.
DC_UP	Configures the dc_name device to UP.
DC_WRITE	Writes the number of bytes (which must be a sector multiple) specified in dc_count to the address specified in dc_buff from the device dc_name.

The possible errors for the system calls and probable causes are as follows:

[EEXIST] This error appears when it is detected that a flaw already exists for block dc_bno (DC_AFLW).

[EFAULT] This error appears when one of the following conditions has occurred:

- The address of the ddcntl structure is out of range.
- The logical device number dc_off is too big (DC_RMAP, DC_MAINT).
- The slice number in dc_count is too big (DC_MAINT).
- The buffer address dc_buff is out of range.

[EINVAL] This error appears when one of the following conditions has occurred:

- The block number to be flawed is out of range (DC_AFLW).
- The slice to be placed into maintenance mode is not DOWN or RONLY (DC_MAINT).
- The SSD channel number was not found (DC_SSDOFF, DC_SSDON).
- Undefined command.

[ENOENT] This error appears when one of the following conditions has occurred:

- The name of the physical device dc_name is not found.
- No spares device is configured (DC_AFLW, DC_DFLW, DC_RFLW).
- The flaw is not found (DC_DFLW, DC_RFLW).

[ENOSPC] This error appears when no spares are left (DC_AFLW, DC_RFLW).

NOTES

Only the super user can use the /dev/disk interface.

CRAY EL systems do not support SSD devices.

FILES

/dev/disk	Physical disk interface file
/usr/include/sys/ddcntl.h	Control structure definition for the ioctl system call
/usr/include/sys/fcntl.h	Structure definition for fcntl
/usr/include/sys/iobuf.h	Structure definition for iobuf
/usr/include/sys/types.h	Data type definition file
cf/conf.*SN*.c	System configuration file (*SN* is the mainframe serial number)

SEE ALSO

`dsk`(4)

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

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

NAME

dm – Kernel to data migration daemon communications interface

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The /dev/dm/mig0 special file contains messages for dmdaemon(8), the data migration daemon. Messages are sent to the data migration daemon when the kernel recognizes the need for a file recall, the removal of an offline file, or the cancellation of a file recall.

The data migration daemon replies to the kernel requests through the same special file. These replies, currently used only for a recall request, indicate either a successful completion of the recall or an error.

The format of the requests and replies on this device are defined in sys/dmkreq.h, as follows:

```
struct dmkr {
        int     kr_magic;         /* for verification purposes   */
        int     kr_req;           /* request                     */
        int     kr_rep;           /* reply                       */
        int     kr_error;         /* error number                */
        int     kr_seq;           /* sequence number             */
        struct  dm_dvino kr_dvi;  /* device/inode of the file    */
        struct  offhdl kr_hdl;    /* file handle of the file     */
} mc_msg;
```

The following are the available ioctl requests, as defined in the sys/dmkreq.h include file:

MIG_DEBUG Sets a debug mode that echoes messages to the system console.

MIG_NBIO Enables or disables nonblocking I/O; currently unused.

MIG_NREAD Returns the number of bytes available to be read.

NOTES

Only one process at a time is permitted to have this device open.

FILES

/dev/dm/mig0 Message file for kernel to data migration communication

SEE ALSO

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

NAME

`dsk` – Disk drive interface

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

The `/dev/dsk` file contains block special files that represent logical disk devices. A *logical disk device* is a collection of blocks on one or more physical disk or other logical disk devices.

The block special files in `/dev/dsk` are the interface to disk devices for the UNICOS file system. They have the major device number of the logical disk driver, `ldd`. See `ldd`(4) and `ldesc`(5).

The block special devices in `/dev/dsk` are made by using the `mknod`(8) command. Each device can reference one logical or physical disk device directly or more than one device indirectly.

The `mknod`(8) command is used as follows to create a logical disk inode:

`mknod` *name* b *major minor* 0 0 *path*

name Name of the logical device.

`b` The device is a block special device.

major Major device number of the concatenated logical disk device driver. The driver is denoted by the name `dev_ldd` and is defined in `/usr/src/uts/c1/cf/devsw.c`.

minor Minor device numbers must be unique among logical disk devices. Major device 0 is reserved for the `/dev/disk` control device (description follows).

`0 0` Placeholders for future use.

path Device path name. Path names for devices are full path names and are limited to 23 characters in length.

Two types of devices can be defined by using the `mknod` command: logical direct devices and logical indirect devices. A *logical direct* device indicates that the logical disk is comprised of exactly one disk slice. The path in the `mknod` parameter represents another block or character special device.

```
mknod /dev/dsk/usr b 34 20 0 0 /dev/pdd/usr
```

A *logical indirect* device indicates that the logical disk is comprised of more than one disk slice. The path in the `mknod` parameter represents a logical descriptor file. See `ldesc`(5).

```
mknod /dev/dsk/usr b 34 21 0 0 /dev/ldd/usr
```

The following `ioctl`(2) system calls are supported through the `/dev/disk` control device.

`ioctl`(*fd*, *cmd*, *arg*)

fd Open file descriptor for `/dev/disk`.

cmd *cmd* can be one of the following parameters:

`DC_ACACHE`	Adds cache to a logical device
`DC_RCACHE`	Removes cache from a logical device
`DC_RMAP`	Returns the `ldmap` structure for the logical device
`DC_SYNCALL`	Flushes the logical disk device cache to disk

arg A pointer to `struct ddctl`. The `ddctl` structure is defined in `sys/ddcntl.h`. The minor device number of the target device is specified in the `dc_off` field.

For a description of the physical characteristics of disk drives for systems with an IOS model E, see `diskspec`(7).

FILES

`/dev/dsk/*`

`/dev/ldd/*`

`/usr/include/sys/ldesc.h`

SEE ALSO

`disk`(4), `ldd`(4), `ldesc`(5), `mdd`(4), `pdd`(4), `ssd`(4), `ssdd`(4)

`diskspec`(7) (available only online) for IOS model E

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

General UNICOS System Administration, Cray Research publication SG–2301

NAME

`err` – Error-logging interface

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

Minor device 0 of the error-logging interface driver, `err`, is the interface between a process and the system's error-record collection routines. The `/dev/error` special file represents the `err` driver. A single process that has super-user permissions may open the driver for reading only. Each `read` operation causes an entire error record to be retrieved. If the `read` request is for less than the length of the record, the record is truncated.

FILES

`/dev/diag`

`/dev/error`

`/dev/MAKE.DEV`

`/usr/include/sys/erec.h`

SEE ALSO

`errfile`(5)

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

NAME

esd – External Semaphore Device Logical-layer Interface

SYNOPSIS

/dev/sfs

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

The /dev/sfs device is used as the interface to the logical layer of the External Semaphore Device (ESD) driver. The logical layer of the ESD driver manages all semaphore lock assignment, heart-beat monitoring, and port table management.

The ioctl(2) system call is used to issue requests to the ESD driver.

The control structure used for the ioctl system call is defined in the sys/esd.h include file, as follows:

```
struct esdctl {
        word    esdf_narg[5];           /* Function arguments */
        word    esdf_reply;             /* Function reply area address */
};

/*
 * used for:    ESDF_ASGNSEMA
 *              ESDF_RELSEMA,
 *              ESDF_SETSEMA,
 *              ESDF_CLRSEMA,
 *              ESDF_TESTSEMA,
 *              ESDF_TSETSEMA,
 *              ESDF_SET_XCLR_SEMA,
 */
#define esdf_fs_id  esdf_narg[0]        /* Semaphore filesys/proc id word 0 */
#define esdf_fs_id1 esdf_narg[1]        /* Semaphore filesys/proc id word 1 */

/*
 * used for:    ESDF_RELSEMA,
 *              ESDF_SETSEMA,
 *              ESDF_CLRSEMA,
 *              ESDF_TSETSEMA,
 *              ESDF_TSETSEMA,
 *              ESDF_SET_XCLR_SEMA,
 */
#define esdf_semano esdf_narg[0]        /* Semaphore number */
#define esdf_fnflags esdf_narg[3]       /* Function flags */
                                        /* ESDF_ID = 0 => use esdf_semano    */
```

```
                                        /* ESDF_ID = 1 => use esdf_fs_id/id1 */

/*
 * used for:    ESDF_TSETSEMA,
 */
#define esdf_timeout esdf_narg[4]       /* Time out value (in clocks) */

/*
 * used for:    ESDF_REPORT
 *              ESDF_READHBEAT
 */
#define esdf_replysz esdf_narg[4]       /* Reply block size (in words) */
```

The first five words of the `esdctl` structure have different meanings, depending on the command being used. The `#define` statements following the definition of the `esdctl` structure provide an easy way of redefining the control words in lieu of using a `union`.

The general method of interfacing to the ESD driver involves passing in the handle for the requested semaphore by name or by semaphore number. When the semaphore name is supplied in `esdf_fs_id` and `esdf_fs_id1` fields of the `request` structure, the `ESDF_ID` flag must be set in the `esdf_fnflags` field. If semaphore number is supplied, the `ESDF_ID` flag must be 0. The `esdf_reply` field of the `esdctl` structure must be initialized to the address of a `reply` structure. The address of the `request` structure is passed in the `ioctl` call.

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

structure esdctl request;
structure esdrep reply;

if (fd = open("/dev/sfs", O_RDONLY)....
      .
      .
request.esdf_reply = (word)&reply;
ioctl(fd, cmd, &request)
```

On return, the `reply` structure contains information filled in based on the command type:

```
struct esdrep {
        word    esdr_semano;            /* semaphore number */
        word    esdr_result;            /* result code (-1 => error) */
        word    esdr_error;             /* error code */
        word    esdr_state;             /* previous/current state */
        word    esdr_last_port;         /* last port holding sema */
        word    esdr_portname;          /* last port (name) holding sema */
        struct  timeval esdr_time;      /* assignment time */
        word    esdr_avail;             /* # available user-assignable sema's */
        word    esdr_used;              /* # used user-assignable sema's */
};
```

A description of the available `ioctl` requests follows:

`ESDF_ASGNSEMA` Assigns a semaphore.

Requires: `esdf_fs_id/esdf_fs_id1`

Returns: `esdr_result`

0 = Success

-1 = Error (for code, see `esdr_error`)

`ESDF_RELSEMA` Releases a semaphore.

Requires: `esdf_fs_id/esdf_fs_id1`, or `esdf_semano`
`esdf_fnflags`

Returns: `esdr_result`

0 = Success

−1 = Error (for code, see `esdr_error`)

`ESDF_SETSEMA` Sets the semaphore to 1.

Requires: `esdf_fs_id/esdf_fs_id1`, or `esdf_semano`
`esdf_fnflags`

Returns: `esdr_result`

0 = Success

−1 = Error (for code, see `esdr_error`)

`ESDF_CLRSEMA` Sets the semaphore to 0.

Requires: `esdf_fs_id/esdf_fs_id1`, or `esdf_semano`
`esdf_fnflags`

Returns: `esdr_result`

0 = Success

		–1 = Error (for code, see `esdr_error`)
`ESDF_TESTSEMA`	Returns the current value of the semaphore.	
	Requires:	`esdf_fs_id/esdf_fs_id1`, or `esdf_semano` `esdf_fnflags`
	Returns:	`esdr_result`
		0 = Success
		–1 = Error (for code, see `esdr_error`)
`ESDF_TSETSEMA`	If the semaphore is currently 0, set it to 1; otherwise, return with a failure.	

	Requires:	`esdf_fs_id/esdf_fs_id1`, or `esdf_semano`
		`esdf_fnflags`
		`esdf_timeout`

		0 = One attempt.
		>0 = Number of clocks periods to wait.
	Returns:	`esdr_result`
		1 = Semaphore already set by this system
		0 = Success
		–1 = Error (for code, see `esdr_error`)
`ESDF_REPORT`	Reports All/Assigned Semaphores	
	Requires:	`esdf_replysz`
`ESDF_READHBEAT`	Read Heart Beat Status	
	Requires:	`esdf_replysz`
`ESDF_SET_XCLR_SEMA`	Sets semaphore unconditionally. Clear, if setting process exits.	
	Requires:	`esdf_fs_id/esdf_fs_id1`, or `esdf_semano` `esdf_fnflags`
	Returns:	`esdr_result`
		1 = Semaphore already set by this system
		0 = Success
		–1 = Error (for code, see `esdr_error`)

EXAMPLES

An example of assigning a semaphore directly follows:

```
/*
 * Initialize the 16-character cluster-unique name
 * to be assigned to this semaphore.  Care should be
 * taken to not use the same conventions as used to
 * identify filesystems.
 */
request.esdf_fs_id  = '01234567';
request.esdf_fs_id1 = '89ABCDEF';

request.esdf_reply = (word)&reply;

if (ioctl(fd, ESDF_ASGNSEMA, &request) < 0) {
      .... error, ioctl refused....
}
```

At this point, the `ioctl` has returned an `esdrep` structure that contains the fields at the `esdf_reply` address:

```
struct esdrep {
  word  esdr_semano;            /* semaphore number          */
  word  esdr_result;            /* result code (-1 => error) */
  word  esdr_error;             /* error code                */
  word  esdr_state;             /* previous/current state    */
  word  esdr_last_port;         /* last port holding sema    */
  word  esdr_portname;          /* last portname holding sema*/
  struct timeval esdr_time;     /* assignment time           */
  word  esdr_avail;             /* # avail user-assignable   */
                                /* sema's                    */
  word  esdr_used;              /* # used user-assgn sema's  */
};

Where:
  esdr_semano             The semaphore # assigned
   esdr_result             0 for success, -1 for some error condition
   esdr_error              If there was an error, the expanded error
                           codes are defined in sys/esd.h
   esdr_state              The 'state' field from the response word
                           returned from the hardware semaphore box
                           after 'clear'ing the semaphore, not very
                           interesting at user-level
   esdr_last_port          The port #    of the assigning system
   esdr_portname           The port name of the assigning system
   esdr_time               The time the assignment occurred
   esdr_avail              The # of hardware semaphores available to
                           be assigned
   esdr_used               The # of hardware semaphores already assigned
```

An example of releasing a semaphore directly follows:

```
/*
 * Indicate which semaphore is to be released
 */
request.esdf_semano  = a_semaphore_#;
request.esdf_fnflags = 0;

request.esdf_reply   = (word)&reply;

if (ioct.l(fd, ESDF_RELSEMA, &request) < 0) {
      .... error, ioctl refused....
}
```

At this point, the `ioctl` has returned an `esdrep` structure in `reply` with the following fields:

```
esdr_semano             The semaphore # released
esdr_result             0 for success, -1 for some error condition
esdr_error              If there was an error, the expanded error
                        codes are defined in sys/esd.h
esdr_state              0
esdr_last_port          The port #    of the releasing system
esdr_portname           The port name of the releasing system
esdr_time               The time the assignment occurred
esdr_avail              The # of hardware semaphores available to
                        be assigned
esdr_used               The # of hardware semaphores already assigned
```

An example of setting a semaphore directly follows:

```
/*
 * Indicate which semaphore is to be set
 */
request.esdf_semano  = a_semaphore_#;
request.esdf_fnflags = 0;

request.esdf_reply   = (word)&reply;

if (ioct.l(fd, ESDF_SETSEMA, &request) < 0) {
      .... error, ioctl refused....
}
```

At this point, the `ioctl` has returned an `esdrep` structure in `reply` with the following fields:

```
esdr_semano          The semaphore # just set
esdr_result          0 for success, -1 for some error condition
esdr_error           If there was an error, the expanded error
                     codes are defined in sys/esd.h
esdr_state           The ‘state’ field from the response word
                     returned from the hardware semaphore box
                     after ‘set’ing the semaphore, indicates
                     whether it was set or clear before the
                     current set operation
esdr_last_port       The port #    of the last system to change the
                     semaphore
esdr_portname        The port name of the last system to change the
                     semaphore
esdr_time            0
esdr_avail           0
esdr_used            0
```

An example of clearing a semaphore directly follows:

```
/*
 * Indicate which semaphore is to be cleared
 */
request.esdf_semano  = a_semaphore_#;
request.esdf_fnflags = 0;

request.esdf_reply   = (word)&reply;

if (ioct.l(fd, ESDF_CLRSEMA, &request) < 0) {
      .... error, ioctl refused....
}
```

At this point, the `ioctl` has returned an `esdrep` structure in `reply` with the following fields:

```
esdr_semano             The semaphore # just cleared
esdr_result             0 for success, -1 for some error condition
esdr_error              If there was an error, the expanded error
                        codes are defined in sys/esd.h
esdr_state              The ‘state’ field from the response word
                        returned from the hardware semaphore box
                        after ‘clear’ing the semaphore, indicates
                        whether it was set or clear before the
                        current clear operation
esdr_last_port          The port #    of the last system to change the
                        semaphore
esdr_portname           The port name of the last system to change the
                        semaphore
esdr_time               0
esdr_avail              0
esdr_used               0
```

NOTE: A clear request is honored only if the semaphore was set by the same system that issues the clear request.

An example of testing a semaphore directly follows:

```
/*
 * Indicate which semaphore is to be tested
 */
request.esdf_fs_id  = ’01234567’;
request.esdf_fs_id1 = ’89ABCDEF’;
request.esdf_fnflags = ESDF_ID;

request.esdf_reply = (word)&reply;

if (ioct.l(fd, ESDF_TESTSEMA, &request) < 0) {
      .... error, ioctl refused....
}
```

At this point, the `ioctl` has returned an `esdrep` structure in `reply` with the following fields:

```
esdr_semano             The semaphore # just cleared
esdr_result             0 for success, -1 for some error condition
esdr_error              If there was an error, the expanded error
                        codes are defined in sys/esd.h
esdr_state              The ‘state’ field from the response word
                        returned from the hardware semaphore box
                        after ‘test’ing the semaphore, indicates
                        whether it was set or clear
esdr_last_port          Port # of the last system to change the
                        semaphore
esdr_portname           Port name of the last system to change the
                        semaphore
esdr_time               0
esdr_avail              0
esdr_used               0
```

NOTES

Only the super user can use the `/dev/sfs` interface.

FILES

`/dev/sfs` External Semaphore Device Logical-layer Interface

SEE ALSO

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

NAME

FDDI – ANSI Fiber Distributed Data Interface

IMPLEMENTATION

Cray PVP systems with an IOS model E

DESCRIPTION

The FDDI interface (or FDDI driver) drives an ANSI standard Fiber Distributed Data Interface (FDDI). Application processes use the FDDI driver by means of the standard UNICOS system calls (that is, close(2), ioctl(2), open(2), read(2), reada(2), write(2), and writea(2)). Each of the special files in a /dev/fddi*n*/* directory represents all of the logical paths on one physical network interface. Each occurrence of a /dev/fddi*n*/* directory represents a physical network interface.

By convention, FDDI file names have the following format:

/dev/fddi*n*/fd*xx*

n Physical interface number

xx Logical path number

All FDDI I/O is raw; that is, the user process is locked in memory as data moves directly between the user buffer and the network. Therefore, user buffers must be word-aligned, and their length must be in 8-byte multiples.

Cray FDDI supports station management (SMT), version 6.2.

FDDI Fundamentals

FDDI is a 100-Mbit/s token-ring network that is used as a high-performance interconnection among computers and peripheral equipment as well as a high-speed backbone network for medium performance local area networks (LANs). FDDI uses fiber-optic technology as its transmission medium and can be configured to support a sustained transfer rate of approximately 80 Mbit/s (10 Mbyte/s). FDDI can interconnect many nodes on a ring distributed over distances of several kilometers in length. FDDI can support rings made up of 1000 physical connections that span a total fiber path length of 200 km. Each point-to-point link that makes up the ring can be a maximum of 2 km in length for fiber-media applications. Other media technologies, such as copper twisted-pair, are being studied but have no standards published as yet. Distance limits and other characteristics of the ring extent are different for these other media types.

An FDDI ring consists of a set of stations logically connected as a serial string of stations and media to form a closed loop. Information is transmitted sequentially from one station to the next; each station regenerating and repeating the information. The station serves as a way of attaching one or more devices to the network to communicate with other devices on the network.

FDDI, as it is defined by the American National Standards Institute, is divided into three layers: physical layer (PL), data link layer (DLL), and station management (SMT). Each layer defines part of FDDI and is kept as independent of the other layers as possible.

The PL is divided into two sublayers: physical medium dependent (PMD) and physical layer protocol (PHY). The PMD defines and characterizes the fiber-optic drivers and receivers, media-dependent encoding requirements, cables and connectors (cable plant), power budgets, optical-bypass provisions, and all other physical hardware-related characteristics. The PHY provides connection between the PMD and DLL. PHY's responsibilities include clock synchronization with incoming code-bit stream, encoding and decoding of code-bit stream into a symbol stream for use by the upper layers, and media conditioning and initializing.

The DLL is divided into two sublayers: media access control (MAC) and logical link control (LLC). The MAC provides fair and deterministic access to the media, address recognition, and generation and verification of frame check sequences. Its primary function is the delivery of frames from station to station, including frame transmission, repetition, and removal. The LLC provides a common peer-to-peer protocol that facilitates the transfer of information and control between any pair of DLL service access points on the FDDI ring.

Station management (SMT) provides the control necessary to manage the processes that occur in all of the FDDI layers such that each station may work cooperatively on the FDDI ring. SMT provides services such as station insertion and removal, station initialization, configuration control, fault isolation and recovery, station isolation, statistics collection, and address administration.

FDDI User Interface

As with any network interfaces on Cray Research systems with an IOS model E systems, when the first logical path is opened on a FDDI interface, the physical channel is configured up. For FDDI, this implies going through physical connection management (PCM) to connect to its adjacent stations, usually referred to as its upstream and downstream neighbors. When the last logical path on a FDDI interface is closed, the physical channel is configured down, which causes the Cray FDDI interface to disconnect from its neighbors.

Before configuring the channel interface, the microprocessor (High Performance microController, or HPC) on the channel adapter must first be downloaded with its microcode. If any I/O operation is tried on the device before the microcode has been downloaded, an error will occur. Automatic configuring of the channel on the first `OPEN` occurs only after the channel adapter has been downloaded.

From the user's perspective, the *frame* is the basic unit of information to and from the network. The maximum frame size on an FDDI network is 4500 bytes, which includes 2 bytes of preamble, 1 byte of start delimiter (SD), 1 byte of frame control (FC), 6 bytes of destination MAC address (DA), 6 bytes of source address (SA), 4478 bytes of Information (INFO), 4 bytes of frame check sequence (FCS), and 2 bytes of ending delimiter/frame status (ED/FS). The FC, DA, and SA fields compose the *MAC header*. The FC byte identifies the frame's type (for example, it identifies the frame as being a token, a MAC frame, a LLC frame, or a SMT frame). MAC addresses used on FDDI are IEEE 48-bit (Ethernet/Canonical) addresses; however, the FDDI MAC standard requires the Ethernet addresses to be in Non-Canonical form (or MSB form) when placed onto the media. FDDI (MSB) form implies that each byte within the 48-bit address is end-for-end bit swapped (0xCC to 0x33). To make this transparent to users, the Cray FDDI interface hardware performs this translation on each frame transmitted or received; therefore, users always see true Ethernet addresses.

On writes, the user must compile a frame made up of the MAC header and the INFO field, but not the FCS or the ED/FS. When the frame is placed on the physical media, the hardware will append the PA, SD, FCS, and ED/FS fields. Therefore, the amount of data that the user is allowed to write is a maximum of 4491 bytes, plus however many pad bytes have been programmed. (Pad bytes are described later in this subsection.) Usually, 3 bytes of pad are before the FC byte when IP is the protocol being run over FDDI. The buffer that holds the frame to be transmitted must be on a Cray word boundary, even though the number of bytes written does not have to be a multiple of Cray words.

On reads, users may post reads of any size, as long as the buffer is aligned on a Cray word boundary. For SMT frames received from the network, if the buffer is large enough, a user will receive the entire FDDI frame from the media, including the 4 bytes of FCS. Users must be aware that these 4 bytes of FCS are in the buffer, and if necessary, subtract 4 from the length to compensate for it. For LCC frames, the cyclic redundancy check (CRC) is stripped off.

Internet protocol (IP) datagrams and address resolution protocol (ARP) requests and replies are sent and received over the FDDI interface. Both of these protocols use the LLC service of FDDI. All LLC services over FDDI use 802.2 LLC (for related documents, see the SEE ALSO section). RFC 1390 defines the specific encapsulation of IP datagrams and ARP requests and replies within an FDDI frame by using 802.2 LLC SNAP. The use of this encapsulation yields a frame in which the beginning of the IP header is not aligned on a Cray word boundary. This is not a desirable situation for the Cray TCP/IP implementation; therefore, to assist the protocol stacks in frame processing, the Cray FDDI hardware can strip and append a set of pad bytes to each frame transmitted or received from the network. The number of pad bytes is programmable from 0 to 7. For IP, the desired number of pad bytes is 3. If padding is enabled, users must compile frames with the pad bytes on writes, and they will receive frames with the pad bytes on reads. The hardware strips off the pad bytes before frame transmission.

Protocols such as ARP must be able to determine the IEEE 48-bit address of the FDDI interface to place this address in ARP replies from other hosts. To determine the MAC address of the Cray FDDI interface, users may do an `ioctl FDC_GET` request (see `FDIO_GETSET`) or an `ioctl COMM_IOC_GETULA` request (see `NETULA`), both of which have versions that the UNICOS kernel can use.

For more information about the `FDDI` standard, see the ANSI documents listed in the SEE ALSO section.

Station Management Assistance

SMT is divided into two major categories: connection management (CMT) and Frame-based Management. CMT is further divided into five separate entities: configuration management (CFM), entity coordination management (ECM), ring management (RMT), link error monitor (LEM), and physical connection management (PCM). The ANSI standard defines how to implement these different parts of SMT; however, the manufacturer of FDDI hardware must determine the implementation of all these different forms of station management.

In the Cray implementation, SMT is a destributed process. The Cray Research mainframe handles all frame-based management in the form of a daemon called SMTD. Connection management (CMT) is handled on the channel adapter itself, using a microprocessor (called the HPC) that runs microcode that manipulates the FDDI chipset hardware and also maintains all of the different software state machines needed to perform CMT.

In both cases, the FDDI driver has hooks to support the efforts of the SMTD and the HPC (for example, the driver maintains a structure for each physical FDDI interface called `smtinfo`). In this structure, data is kept on behalf of the SMTD. The data consists of such things as upstream and downstream neighbor addresses, MAC availability information, results of the Duplicate Addresses Test that the RMT performs in the HPC, and a part of the SMT time-stamp value. To access these values, the SMT uses the `ioctl` interface.

The mainframe also must have some control over the microcode that is running in the HPC on the channel adapter. It needs this for two reasons. First, the SMTD must be able to retrieve information from both the state machines and the physical FDDI chipset on the channel adapter to respond to SMT frames received from other stations on the FDDI network. To do this, an interface that makes the HPC look like a set of readable and writable registers to the mainframe was designed. Using this interface (which also uses `ioctl` requests), the SMTD can read or write some of the key registers on the channel adapter at virtually any time.

Second, when certain events occur on the channel adapter or the FDDI ring itself, the HPC notifies the mainframe by sending an *unsolicited interrupt*. Several events can cause one of these interrupts; some events need immediate action by the driver, and others do not. Some examples of events that cause interrupts to the mainframe are as follows:

- Changes in MAC availability
- The ring recovering from a TRACE (which is an FDDI term for media reconfiguration)
- The overflow of a 32-bit SMT time-stamp value

When these events occur, the FDDI driver takes the necessary action(s). There are times when the mainframe must force the CMT state machines to a given state (for example, if the mainframe wants to disconnect from the FDDI network, it must tell the HPC's PCM code to disconnect). It does this by use of another interface, which is similar to the register read-write interface, called the *HPC signal interface*. Again, by using `ioctl` requests, the mainframe can send a signal to the HPC to perform some action. Examples of these signals are Duplicate Address Test Failed (sent by the SMTD), `EC_CONNECT` (which causes a port to connect to its neighbor), and `EC_DISCONNECT` (which causes a port to disconnect from its neighbor).

Parameter File

At boot time, the FDDI driver is configured by using a parameter file. The following example shows the syntax for parameters entered in the parameter file:

```
2 fdmaxdevs;                        /* max no. of FDDI interfaces */
16 fdmaxpaths;                      /* max no. of lpaths per interface */
fddev 0 {                           /* first interface */
        treq 10;                    /* FDDI TREQ in milliseconds */
        padcnt 3;                   /* no. of pad bytes */
        maxwrt 10;                  /* max no. of write requests to IOS */
        maxrd 10;                   /* max no. of read requests to IOS */
        iopath {
                cluster 0;
                eiop 1;
                channel 034;
        }
        logical path 0 {
                rft SMT;           /* want to receive SMT frames */
                read timeout 20;   /* read time-out in seconds */
        }
        logical path 1 {
                rft ALL;           /* want to receive ALL frame types */
                read timeout 10;
        }
        logical path 5 {
                rft LLC;           /* want to receive LLC frames */
                read timeout 60;
        }
}
fddev 1 {                          /* second interface */
        treq 30;
        padcnt 0;
        maxwrt 5;
        maxrd 5;
        iopath {
                cluster 0;
                eiop 1;
                channel 036;
        }
        logical path 0 {
                read timeout 30;
        }
        logical path 1 {
                read timeout 15;
        }
}
```

The only mandatory parameter for FDDI is `iopath`. Without this parameter, the driver does not know of the physical location of the FDDI interface. If you omit this parameter, an open on any logical path on that device will receive a `FDER_NOCHAN` error.

FDDI `ioctl` Requests

Several `ioctl` requests are available, and they have the following format:

```
#include <sys/netdev.h>
#include <sys/fd.h>
#include <sys/fdsys.h>

ioctl (fildes, command, arg)
int fildes;      /* file descriptor returned on open */
int command;     /* request code (FDC_XXX in sys/fd.h) */
char *arg;       /* fdioreq, commstreq, or netula */
```

When an `ioctl` request is performed, the *arg* argument of the system call must point to either a `fdioreq` structure or a `commstreq` structure. Each of these structures have pointers to buffers that hold the actual `ioctl` data. These buffers must be of sufficient size to hold the data requested. If the request will pass back multiple structures of a given type, the buffer must be large enough to hold all instances of the requested structure.

The following shows the format of the structures used in the `ioctl` request. You can find those structures that are not shown in the various header files listed previously.

```
struct macaddr {
        uint         :16,  /* unused */
                ieee :48;  /* IEEE (Canonical) form */
        uint         :16,  /* unused */
                fddi :48;  /* FDDI (MSB) form */
};

struct netula {
        long    addr;      /* 48-bit address (Canonical form) */
};

struct commstreq {
        int     sfunc;     /* status request subfunction */
        char    *sbuf;     /* status buffer pointer */
        int     dev;       /* device number (-1 means all devices) */
        int     lpath;     /* logical path (-1 means all paths) */
        uint    slen;      /* status buffer length (bytes) */
        uint    epoch;     /* incremented every configuration change */
};
```

```
struct fdioreq {
        char    *buf;       /* buffer pointer */
        int     len;        /* buffer length */
        int     param;      /* parameter */
};

struct fdio_echo {
        char    data [FD_MAXECHO];
};

struct fdio_loadmicro {
        char    binary [FD_MAXLOAD];

};

struc fdio_dumpsm {
       char    data [FD_MAXDUMPSM]

};
struct fdio_getset {
        int     ieee_mac;   /* fd->mac.ieee (RO) */
        int     fddi_mac;   /* fd->mac.fddi (RO) */
        int     errno;      /* lp->errno (RO) */
        int     err;        /* lp->err (RO) */
        int     des;        /* lp->des (RO) */
        int     fsw;        /* lp->fsw (RO) */
        int     TREQ;       /* fd->TREQ (RW) */
        int     cc;         /* fd->cc (RW) */
        int     padcnt;     /* fd->padcnt (RW) */
        int     maxwrt;     /* fd->maxwrt (RW) */
        int     maxrd;      /* fd->maxrd (RW) */
        int     opt;        /* lp->opt (RW) */
        int     rft;        /* lp->rft (RW) */
        int     rtmo;       /* lp->rtmo (RW) */
};

struct fdio_smt_timestamp {
        int     hi_32;      /* Upper 32 bits maintained by driver */
};

struct fdio_dad_results {
        int     results;      /* results of Duplicate Addr Test */

};
```

```
struct fdio_mac_neighbors {
        struct  macaddr una; /* upstream neighbor */
        struct  macaddr dna; /* downstream neighbor */
};

struct fdio_hpc_reg_data {
        int     size;     /* size of access (none, byte, word, long) */
                             /* F_HPC_SIZE_NONE, F_HPC_SIZE_BYTE */
                             /* F_HPC_SIZE_WORD, F_HPC_SIZE_LONG */
        int     page;        /* memory page address */
                             /* FD_HPC_PAGE_SMT, FD_HPC_PAGE_RMT */
                             /* FD_HPC_PAGE_PORT1, FD_HPC_PAGE_PORT2 */
                             /* FD_HPC_PAGE_BMAC, FD_HPC_PAGE_PLAYER1 */
                             /* FD_HPC_PAGE_PLAYER2, FD_HPC_PAGE_DIAG */
        int     addr;        /* HPC register address (0x00 - 0xff) */
        int     data;        /* read or write data (right justified) */
};

struct fdio_signal_hpc {
        int     signo;       /* signal number to HPC */
                             /* F_HPC_SIG_RMT_DAD_FAIL */
                             /* F_HPC_SIG_RMT_DAD_PASS */
                             /* F_HPC_SIG_RMT_EC_DISCONNECT */
                             /* F_HPC_SIG_RMT_EC_CONNECT */
                             /* F_HPC_SIG_RMT_EC_PTPASS */
                             /* F_HPC_SIG_RMT_SYS_RESET */
};

struct fdio_hpc_info {
        uchar   smt_00[2];    /* Working Register I */
        uchar   smt_02[2];    /* Working Register J */
        uchar   smt_04[2];    /* Working Register P */
        uchar   smt_06;       /* Microcode Revision */
        uchar   smt_07;       /* Hardware Configuration */
        uchar   smt_08;       /* SMT State */
        uchar   smt_09;       /* SMT Configuration */
        uchar   smt_0A;       /* Interrupt Summary */
        uchar   smt_0B;       /* Interrupt Mask */
        uchar   smt_0C[2];    /* SMT Events */
        uchar   smt_0E[2];    /* SMT Event Mask */
        uchar   smt_10[4];    /* SMT Timestamp */
        uchar   smt_14[2];    /* SYSTIM Counter */
        uchar   smt_16;       /* Timer Events */
        uchar   smt_17;       /* Timer Event Mask */
```

```
uchar   rmt_00;        /* RMT State */
uchar   rmt_01;        /* RMT Status */
uchar   rmt_02[2];     /* RMT Timer */
uchar   rmt_04[2];     /* RMT Events */
uchar   rmt_06[2];     /* RMT Event Mask */
uchar   rmt_08[2];     /* Reserved word */
uchar   rmt_0A[2];     /* RTT - Restricted Token Timeout */
uchar   rmt_0C[2];     /* T_request value */
uchar   mac_0E[2];     /* Late_Ct (Late Count) */
uchar   mac_10[4];     /* Token_Ct (Token Count) */
uchar   mac_14[4];     /* TX_Ct (Transmit Count) */
uchar   mac_18[4];     /* NotCopied_Ct (Not Copied Count) */
uchar   mac_1C[4];     /* Frame_Ct (Frame Copied Count) */
uchar   mac_20[4];     /* Lost_Ct (Lost Frame Count) */
uchar   mac_24[4];     /* Error_Ct (Error Isolated Count) */
uchar   mac_28[4];     /* RX_Ct (Receive Count) */
uchar   mac_2C[4];     /* Ring_Ct (Ring Recovery Count) */
uchar   mac_30[4];     /* TVX_Ct (TVX Expiration Count) */
uchar   mac_34[4];     /* Latency_Ct (Ring Latency Count) */
uchar   port1_00;      /* Port 1 - PCM state */
uchar   port1_01;      /* Port 1 - PCM-PSC Index */
uchar   port1_02;      /* Port 1 - BREAK Count */
uchar   port1_03;      /* Port 1 - PCM Index at last BREAK */
uchar   port1_04[2];   /* Port 1 - Transition Table Pointer */
uchar   port1_06[2];   /* Port 1 - Current Line State */
uchar   port1_08[2];   /* Port 1 - Reserved Word */
uchar   port1_0A[2];   /* Port 1 - Start Value of TPC Counter */
uchar   port1_0C[2];   /* Port 1 - TCP Counter */
uchar   port1_0E[2];   /* Port 1 - Address of TPC Service Routine */
uchar   port1_10[2];   /* Port 1 - PSC Connection Policy */
uchar   port1_12[2];   /* Port 1 - PSC Rval */
uchar   port1_14[2];   /* Port 1 - PSC Tval */
uchar   port1_16;      /* Port 1 - PSC Status (Neighbor porttype) */
uchar   port1_17;      /* Port 1 - LCT Fail */
uchar   port1_18[2];   /* Port 1 - CEM Policy */
uchar   port1_1A;      /* Port 1 - CEM State */
uchar   port1_1B;      /* Port 1 - PLAYER Configuration Reg value */
uchar   port1_1C;      /* Port 1 - LEM Reject Count */
uchar   port1_1D;      /* Port 1 - LER Estimate */
uchar   port1_1E;      /* Port 1 - LER Alarm */
uchar   port1_1F;      /* Port 1 - LER Cutoff */
uchar   port1_20[4];   /* Port 1 - LEM Count */
uchar   port1_24[4];   /* Port 1 - Elasticity Buffer Error Count */
uchar   port1_28[4];   /* Port 1 - LER Average */
```

```
uchar   port1_2C[4];  /* Port 1 - LER Delta */
uchar   port1_30;     /* Port 1 - ECM State */
uchar   port1_31;     /* Port 1 - ECM Path */
uchar   port1_32[2];  /* Port 1 - Reserved Word */
uchar   port1_34[2];  /* Port 1 - PORT Events */
uchar   port1_36[2];  /* Port 1 - PORT Event Mask */
uchar   port2_00;     /* Port 2 - PCM state */
uchar   port2_01;     /* Port 2 - PCM-PSC Index */
uchar   port2_02;     /* Port 2 - BREAK Count */
uchar   port2_03;     /* Port 2 - PCM Index at last BREAK */
uchar   port2_04[2];  /* Port 2 - Transition Table Pointer */
uchar   port2_06[2];  /* Port 2 - Current Line State */
uchar   port2_08[2];  /* Port 2 - Reserved Word */
uchar   port2_0A[2];  /* Port 2 - Start Value of TPC Counter */
uchar   port2_0C[2];  /* Port 2 - TCP Counter */
uchar   port2_0E[2];  /* Port 2 - Address of TPC Service Routine */
uchar   port2_10[2];  /* Port 2 - PSC Connection Policy */
uchar   port2_12[2];  /* Port 2 - PSC Rval */
uchar   port2_14[2];  /* Port 2 - PSC Tval */
uchar   port2_16;     /* Port 2 - PSC Status (Neighbor porttype) */
uchar   port2_17;     /* Port 2 - LCT Fail */
uchar   port2_18[2];  /* Port 2 - CEM Policy */
uchar   port2_1A;     /* Port 2 - CEM State */
uchar   port2_1B;     /* Port 2 - PLAYER Configuration Reg value */
uchar   port2_1C;     /* Port 2 - LEM Reject Count */
uchar   port2_1D;     /* Port 2 - LER Estimate */
uchar   port2_1E;     /* Port 2 - LER Alarm */
uchar   port2_1F;     /* Port 2 - LER Cutoff */
uchar   port2_20[4];  /* Port 2 - LEM Count */
uchar   port2_24[4];  /* Port 2 - Elasticity Buffer Error Count */
uchar   port2_28[4];  /* Port 2 - LER Average */
uchar   port2_2C[4];  /* Port 2 - LER Delta */
uchar   port2_30;     /* Port 2 - ECM State */
uchar   port2_31;     /* Port 2 - ECM Path */
uchar   port2_32[2];  /* Port 2 - Reserved Word */
uchar   port2_34[2];  /* Port 2 - PORT Events */
uchar   port2_36[2];  /* Port 2 - PORT Event Mask */
uchar   bmac_87;      /* THSH1 - async priority 1 */
uchar   bmac_8B;      /* THSH2 - async priority 2 */
uchar   bmac_8F;      /* THSH2 - async priority 3 */
uchar   bmac_93;      /* T_max */
uchar   bmac_97;      /* TVX_value */
uchar   bmac_98[4];   /* T_negotiated */
uchar   cfm_state;    /* station CFM state */
```

;

The following table shows each of the requests and the structure formats used:

Request	`ioctl arg` parameter	`fdioreq.buf` or `commstreq.sbuf`
COMM_IOC_CDSTATS	commstreq	N/A
COMM_IOC_CLSTATS	commstreq	N/A
COMM_IOC_DSTATS	commstreq	commstat
COMM_IOC_LSTATS	commstreq	commstat
COMM_IOC_STATS	commstreq	commstat
COMM_IOC__GETULA	netula	N/A
COMM_IOC__KGETULA	netula	N/A
FDC_GET	fdioreq	fdio_getset
FDC_SET	fdioreq	fdio_getset
FDC_KGET	fdioreq	fdio_getset
FDC_KSET	fdioreq	fdio_getset
FDC_CDSTATS	commstreq	N/A
FDC_CLSTATS	commstreq	N/A
FDC_DSTATS	commstreq	commstat
FDC_LSTATS	commstreq	commstat
FDC_STATS	commstreq	commstat
FDC_ECHO	fdioreq	fdio_echo
FDC_ECHOSINK	fdioreq	fdio_echo
FDC_CLRDLF	fdioreq	N/A
FDC_SETDLF	fdioreq	N/A
FDC_LOADMICRO	fdioreq	fdio_loadmicro
FDC_DUMPSM	fdioreq	fdio_dumpsm
FDC_DSTRUCT	commstreq	fd_dev
FDC_LSTRUCT	commstreq	fd_lp
FDC_GETVARS	commstreq	fd_vars
FDC_SET_LLC_AV	none	N/A
FDC_CLR_LLC_AV	none	N/A
FDC_XCHG_DAD	fdioreq	fdio_dad_results
FDC_SET_MACNBRS	fdioreq	fdio_mac_neighbors
FDC_GET_MACNBRS	fdioreq	fdio_mac_neighbors
FDC_GET_DAD	fdioreq	fdio_dad_results
FDC_GET_HPC	fdioreq	fdio_hpc_info
FDC_OR_HPC_REG	fdioreq	fdio_hpc_reg_data
FDC_AND_HPC_REG	fdioreq	fdio_hpc_reg_data
FDC_RD_HPC_REG	fdioreq	fdio_hpc_reg_data
FDC_WR_HPC_REG	fdioreq	fdio_hpc_reg_data

Request	ioctl arg parameter	fdioreq.buf or commstreq.sbuf
FDC_SIGNAL_HPC	fdioreq	fdio_signal_hpc
FDC_RD_SMTTIME	fdioreq	fdio_smt_timestamp

The valid ioctl requests are as follows:

COMM_IOC_CDSTATS
: Clears the statistics associated with the device(s) specified by commstreq.dev. If commstreq.dev is a –1, all configured devices will be cleared.

COMM_IOC_CLSTATS
: Clears the statistics associated with the logical path(s) specified by commstreq.dev and commstreq.lpath. If commstreq.dev is a –1 and commstreq.lpath is a –1, all configured paths on all configured devices will be cleared.

COMM_IOC_DSTATS
: Returns the statistics for the device(s) specified by the commstreq.dev parameter. The format of the statistics returned is that of a commstat structure. If commstreq.dev is a –1, statistics for all configured devices will be returned.

COMM_IOC_LSTATS
: Returns the statistics for the logical path(s) specified by the commstreq.lpath parameter. The format of the statistics returned is that of a commstat structure. If commstreq.dev is a –1 and commstreq.lpath is a –1, statistics for all configured logical paths on all configured devices will be returned.

COMM_IOC_STATS
: Returns both device and logical path statistics associated with the device(s) and logical path(s) specified by commstreq.dev and commstreq.lpath. This is the same as FDC_DSTATS and FDC_LSTATS combined.

COMM_IOC_GETULA
: Gets the value of the IEEE Universal LAN address.

COMM_IOC_KGETULA
: Same as FDC_GETULA except that the kernel uses it.

FDC_GET
: Gets the current driver parameter settings and stores them in the fdio_getset structure to which fdioreq.buf points.

FDC_SET
: Sets driver parameters from the `fdio_getset` structure to which `fdioreq.buf` points. All driver parameters that are changed by this `ioctl` request reset to their boot-time values when the logical path is closed. Parameters that apply to the device as a whole, such as `TREQ`, are reset to their boot-time values when the last logical path is closed on the device.

FDC_KGET
: Same as `FDC_GET` except that the kernel uses it.

FDC_KSET
: Same as `FDC_SET` except that the kernel uses it.

FDC_CDSTATS
: Same as `COMM_IOC_CDSTATS`.

FDC_CLSTATS
: Same as `COMM_IOC_CLSTATS`.

FDC_DSTATS
: Same as `COMM_IOC_DSTATS`.

FDC_LSTATS
: Same as `COMM_IOC_LSTATS`.

FDC_STATS
: Same as `COMM_IOC_STATS`.

FDC_ECHO
: Sends the data to which `fdioreq.buf` points to the IOS I/O buffer. This is a simulated write operation that does not activate the channel. To read data back, post a read command on the device.

FDC_ECHOSINK
: Sends the data to which `fdioreq.buf` points to the IOS I/O buffer. This is a simulated write operation that does not activate the channel. Data cannot be read back by posting a read command on the device.

FDC_CLRDLF
: Clears the internal microcode download flag in the driver. When the flag is clear, the driver does not allow any I/O to be performed on the device. Also, when the flag is clear, the channel is not automatically configured on the first open.

FDC_SETDLF
: Sets the internal microcode download flag in the driver. When the flag is set, the driver allows I/O to be performed on the device. Also, when the flag is set, the channel is automatically configured on the first open.

`FDC_LOADMICRO`
: Sends the binary data to which `fdioreq.buf` points in the IOS I/O buffer. The IOP takes this binary data and loads it into the FCA-1's shared memory at the address specified in the binary data itself. After the reset signal is inactivated to the FCA-1, the HPC processor will begin to execute the code that was just downloaded.

`FDC_DUMPSM`
: Returns the contents of FCA-1 shared memory. The `fdioreq.param` specifies the address of the shared memory to be returned.

`FDC_DSTRUCT`
: Returns the contents of the `fd_dev` structures for the device(s) specified by `commstreq.dev`. This structure is the internal driver structure used for controlling each FDDI interface.

`FDC_LSTRUCT`
: Returns the contents of the `fd_lp` structures for the logical path(s) specified by `commstreq.dev` and `commstreq.lpath`. This structure is the internal driver structure used for controlling each logical path on each of the FDDI interfaces.

`FDC_GETVARS`
: Returns the contents of the `fd_vars` structure, which contains the maximum number of FDDI devices and logical paths.

`FDC_SET_LLC_AV`
: Sets the logical link control (LLC) available flag, `fd_dev.smt.llc_available`, for this device.

`FDC_CLR_LLC_AV`
: Clears the LLC available flag, `fd_dev.smt.llc_available`, for this device.

`FDC_XCHG_DAD`
: Exchanges the new results of the Duplicate Address Test with the current results that are stored in the Driver Device table. If the results are different, the driver will notify the HPC on the channel adapter of the new results.

`FDC_SET_MACNBRS`
: Sets the upstream and downstream MAC neighbor addresses from the `fdio_mac_neighbors` structure to which `fdioreq.buf` points.

`FDC_GET_MACNBRS`
: Returns the upstream and downstream MAC neighbor addresses. The format of the data returned is in the form of the `fdio_mac_neighbors` structure.

`FDC_GET_DAD`
: Returns the results of the Duplicate Address Test. The format of the data returned is in the form of the `fdio_dad_results` structure.

FDC_GET_HPC
: Returns the register information from the HPC on the channel adapter. The format of the data returned is in the form of the `fdio_hpc_info` structure.

FDC_OR_HPC_REG
: Logically ORs the contents of a particular HPC register on the channel adapter with the specified value. The format of the data to OR is in the form of the `fdio_hpc_reg_data` structure.

FDC_AND_HPC_REG
: Logically ANDs the contents of a particular HPC register on the channel adapter with the specified value. The format of the data to AND is in the form of the `fdio_hpc_reg_data` structure.

FDC_RD_HPC_REG
: Returns the contents of a particular HPC register on the channel adapter. The format of the data returned is in the form of the `fdio_hpc_reg_data` structure.

FDC_WR_HPC_REG
: Sets the contents of a particular HPC register on the channel adapter to the specified value. The format of the data to write is in the form of the `fdio_hpc_reg_data` structure.

FDC_SIGNAL_HPC
: Sends the specified signal number to the HPC on the channel adapter. The format of the data is in the form of the `fdio_signal_hpc` structure.

FDC_RD_SMTTIME
: Returns the high-order 32 bits of the 64-bit SMT time stamp that the driver maintains by the low-order 32 bits. The HPC on the channel adapter maintains the low-order bits, and they are available in the `fdio_hpc_info` structure. The format of the data is in the form of the `fdio_smt_timestamp` structure.

The `fdio_getset` structure contains the following fields:

`int ieee_mac`
: MAC address; IEEE (Canonical/Ethernet) form. (Ignored on `FDC_SET` and `FDC_KSET`.)

`int fddi_mac`
: MAC address; FDDI form. (Ignored on `FDC_SET` and `FDC_KSET`.)

`int errno`
: Error code returned to user (`errno`). (Ignored on `FDC_SET` and `FDC_KSET`.)

`int err`
: FDDI error code. (Ignored on `FDC_SET` and `FDC_KSET`.)

`int des`
: Detailed error status code from IOS. (Ignored on `FDC_SET` and `FDC_KSET`.)

`int fsw`
: Frame status word from last frame read on this logical path. (Ignored on `FDC_SET` and `FDC_KSET`.)

`int TREQ`
: `TREQ` value for this interface. `TREQ` is the value that is sent on all Claim frames from the `FDDI BMAC`. This is the requested value for the token rotation timer. (Default is 167 ms.)

`int cc`
: Copy criteria mask for this interface. For the specific values for this mask, see `sys/epackf.h`. (Default is `LLC` and `SMT`.)

`int padcnt`
: Pad count (default is 3 bytes) for this interface. This is the number of bytes that are removed from the start of each frame transmitted and inserted at the start of each frame received. This padding is required for protocols such as TCP/IP, which needs the IP header word aligned within the FDDI frame.

`int maxwrt`
: Maximum number of write requests (default is 10) allowed to IOS. After this many write requests are pending in the IOS, the driver queues up any further write requests.

`int maxrd`
: Maximum number of read requests (default is 10) that can be sent to the IOS. After this number of read requests is pending in the IOS, the driver queues any additional read requests.

`int opt`
: Options (default is `NONE`) for this logical path. Currently, two options (`NFRCHK` and `NERRLOG`) are defined for logical paths. The `NFRCHK` option causes the driver to bypass frame validity checking on write operations. This option can be useful for diagnostics. The `NERRLOG` option causes the driver to not log errors that occur on this logical path to the system error log. This option is useful for diagnostics that are causing errors intentionally and want to avoid having a record of those errors. For the bit definitions of each of these options, see the `FDLO_XXXX`, defined in `sys/fd.h`.

`int rft`
: Receive frame type mask (default is `NONE`) for this logical path. This field defines the types of frames to be received by this logical path. Frame types can be SMT frames, LLC frames, or any combination of frame types. However, any particular frame type can be registered to be received only by a single-logical path. If a logical path tries to register to receive a frame type that is already being received by another path, an error will occur. For the specific values for this mask, see the `EFOP_RFT_XXXX`, defined in `sys/epackf.h`.

`int rtmo`
: Read time-out value (in seconds) for this logical path; default is 60 seconds.

EXIT STATUS

The FDDI driver returns one of the following error codes in `errno` on an error. To obtain a more specific error code information, use a `FDC_GET ioctl` request and examine the `err` and `des` fields. For the mapping of the specific error codes to user error codes, see the table that follows.

The error codes and their meanings are as follows:

EBUSY The specified logical path is in use.

EFAULT An argument to a `ioctl` request is not valid.

EINVAL The driver has detected a parameter error.

EIO This error can be caused when a fatal I/O error occurs in the IOP, the FDDI MAC or LLC services are not available on a write, a frame type that was not valid was received from the network, the error bit is set in the frame status in a received frame on a read, or the actual transfer length does not equal the request transfer length on a write or read.

ELATE A request timed out.

EPERM The driver has detected an operation that requires super-user privileges by a user that does not have those privileges.

ENXIO The specified device or logical path does not exist or is not in an operational state.

The mainframe driver returns the following specific error codes:

FDER_BADDR
A `b_waddr` value that is not valid in the *buf* (bp) structure was detected on a read or write.

FDER_BADHPCADDR
A `FDC_OR_HPC_REG`, `FDC_AND_HPC_REG`, `FDC_RD_HPC_REG`, or `FDC_WR_HPC_REG` was issued with an HPC address that is not valid. The address must be less than or equal to 0x00ff.

FDER_BADHPCPAGE
A `FDC_OR_HPC_REG`, `FDC_AND_HPC_REG`, `FDC_RD_HPC_REG`, or `FDC_WR_HPC_REG` was issued with a HPC page address that is not valid. The page must be less than or equal to 0x07.

FDER_BADHPCSIZE
A `FDC_OR_HPC_REG`, `FDC_AND_HPC_REG`, `FDC_RD_HPC_REG`, or `FDC_WR_HPC_REG` was issued with a HPC size that is not valid. The size must be byte, word, or long.

FDER_BADIOFLAG
The `u_io` flag is not set for the user or the kernel making an `ioctl` request.

FDER_BCOUNT
A `b_count` value that is not valid in the *buf* (bp) structure was detected on a read or write.

FDER_BUSY
An open operation is tried on a logical path that is already open.

FDER_CLSNOPEN
A close operation is tried on a logical path that is not open.

FDER_CONFINGDN
An open operation is tried on a device that is in the process of being configured down.

FDER_CONFUPER
An open operation is tried on a device that did not configure up successfully.

`FDER_COPYIN`
: An error occurred when copying data from the user to the kernel.

`FDER_COPYOUT`
: An error occurred when copying data from the kernel to the user.

`FDER_ESET`
: The error bit is set in the received frame.

`FDER_HALTED`
: The I/O was halted.

`FDER_IOCPARAM`
: A parameter that was not valid was detected on an `ioctl` request.

`FDER_IOCREQUEST`
: A `ioctl` request that was not valid was made.

`FDER_LLC`
: An unsupported 802.2 LLC is detected.

`FDER_LLCNAVAIL`
: The LLC services are not available.

`FDER_MACNAVAIL`
: The MAC services are not available.

`FDER_NOCHAN`
: An open operation is tried on a device that has no physical channel defined in the configuration.

`FDER_NOPEN`
: A operation is tried on a logical path that is not open.

`FDER_NORFT`
: A read is posted on this path, but the path has not yet registered to receive any frame types.

`FDER_NOTZUP`
: A nonsuper user tried an `FDC_SET`.

`FDER_NULLDA`
: A null destination address (DA) in the FDDI frame was detected on a write.

`FDER_OK`
: No error (binary 0).

`FDER_PACKLEN`
: The length of an F-packet received from the IOS was incorrect.

`FDER_PACKOUT`
: An error was detected when sending an F-packet to the IOS.

FDER_RANGE
A device or logical path index is out of range. Typically, error this occurs when the minor device number is created incorrectly.

FDER_RCVFC
A frame type that was not valid was received from the network.

FDER_RFTINUSE
The frame type that you want to register for a path is already registered to another logical path.

FDER_WRITEFC
A frame control (FC) byte that is not valid in the FDDI frame was detected on a write.

FDER_XNFRLEN
The actual transfer length does not equal the requested transfer length.

FDER_UNSOLICITED
An unsolicited error has been received from the EIOP. This error may be asynchronous with the operation that the mainframe is currently performing on the channel.

FDER_NOTLOADED
An attempt was made to open a logical path or to perform a read or write on a logical path before the channel adapter has been downloaded with microcode.

FDER_ALRDYLOADED
An attempt was made to download the channel adapter with the download flag already set, meaning the adapter had been already loaded.

The IOS driver returns the following specific error codes:

F_RSP_ACT_CRE8
Cannot create needed activity.

F_RSP_BADCHN
Channel number is not valid.

F_RSP_BADFCA1
FCA1 mode is not valid.

F_RSP_BADLEN
Requested transfer length is not valid.

F_RSP_BADPATH
Logical path is not valid.

F_RSP_BADREQ
Request code is not valid.

F_RSP_BADTMO
Supplied time-out value is not valid.

F_RSP_BAD_TYPE
: Bad packet type.

F_RSP_CBREL_BAD
: CB_Release error.

F_RSP_CBRES_BAD
: CB_Reserve error.

F_RSP_CBNAVL
: Channel buffer not available.

F_RSP_CH_DOWN
: Channel pair not configured up.

F_RSP_CH_INITING
: Initialization of channel pair already in progress.

F_RSP_CH_TERMING
: Termination of channel pair already in progress.

F_RSP_CH_UP
: Channel pair already configured up.

F_RSP_CLOSED
: Request aborted because of `CLOSE PATH` request.

F_RSP_DVRTERM
: Driver terminated.

F_RSP_FCA1_BAD
: `FCA1` hardware information is not valid.

F_RSP_HALTED
: Request aborted because of `HALT I/O` request.

F_RSP_HALT_FAIL
: Cannot `HALT I/O` in a driver activity.

F_RSP_IOBMEM
: IOB memory not available.

F_RSP_LOCMEM
: Local memory not available.

F_RSP_LMIO_ADR
: Bad channel buffer address parameter in `LMIO` request.

F_RSP_LMIO_DIR
: Bad transfer direction parameter in `LMIO` request.

F_RSP_LMIO_HDWR
: Hardware error detected on `LMIO` attempt.

F_RSP_LMIO_LEN
Bad word length parameter in LMIO request.

F_RSP_LMIO_ORD
Bad ordinal parameter in LMIO request.

F_RSP_OK
No error was detected.

F_RSP_OVERRUN
Transferred more data than expected.

F_RSP_ABORT_O
Output I/O buffer to FIFO abort.

F_RSP_BUFMEM_O
Output I/O buffer single-or double-bit error.

F_RSP_PAR_FETCH
Parity error during HPC fetch memory reference.

F_RSP_IFCA1TMO
Input FCA-1 channel time-out.

F_RSP_HPC_DOWNLOAD
HPC download failed.

F_RSP_HPC_BADRESP
HPC sent bad response to mailbox command.

F_RSP_HPC_TMO
Time-out occurred on a HPC mailbox request.

F_RSP_PARITY_O
Output FIFO data/control/buffer parity error detected.

F_RSP_PARITY_I
Input FIFO data/control/buffer parity error detected.

F_RSP_PATHCLO
No open connection for this logical path.

F_RSP_PATHOPN
Connection already open for this logical path.

F_RSP_PKTLEN
Request packet length is not valid.

F_RSP_RD_PKT_TMO
Read request packet timed out.

F_RSP_RELMEM
RELMEM request failed.

F_RSP_TERM_FAIL
Cannot terminate all driver activities.

F_RSP_TIMER_PAR
Bad parameter on TIMER call.

F_RSP_TIMER_QUED
Start a timer already on RTC queue.

F_RSP_TIMER_UNKNOWN
An unknown timer failure.

F_RSP_TMIO_ADR
Bad channel buffer address parameter in TMIO request.

F_RSP_TMIO_DIR
Bad transfer direction parameter in TMIO request.

F_RSP_TMIO_HDWR
Hardware error detected on TMIO attempt.

F_RSP_TMIO_LEN
Bad word length parameter in TMIO request.

F_RSP_TMIO_NAVL
Target memory channel not available.

F_RSP_TMIO_ORD
Bad ordinal parameter in TMIO request.

The following shows the mapping of specific error codes and user error codes:

Specific Error Code	**User Error Code (errno)**
FDER_BADDR	EINVAL
FDER_BADHPCADDR	EINVAL
FDER_BADIOFLAG	EINVAL
FDER_BCOUNT	EINVAL
FDER_BUSY	EBUSY
FDER_CLSNOPEN	ENXIO
FDER_CONFINGDN	ENXIO
FDER_CONFUPER	ENXIO
FDER_COPYIN	EFAULT
FDER_COPYOUT	EFAULT
FDER_ESET	EIO
FDER_HALTED	EIO

FDER_IOCPARAM	EINVAL
FDER_LLCNAVAIL	ENXIO
FDER_MACNAVAIL	ENXIO
FDER_NOCHAN	ENXIO
FDER_NOPEN	ENXIO
FDER_NORFT	EIO
FDER_NOTZUP	EPERM
FDER_NULLDA	EINVAL
FDER_PACKLEN	EIO
FDER_PACKOUT	EIO
FDER_RCVFC	EIO
FDER_RFTINUSE	EIO
FDER_WRITEFC	EINVAL
FDER_XNFRLEN	EIO
F_RSP_ACT_CRE8	EIO
F_RSP_BADCHN	EINVAL
F_RSP_BADFCA1	EINVAL
F_RSP_BADLEN	EINVAL
F_RSP_BADPATH	EINVAL
F_RSP_BADREQ	EINVAL
F_RSP_BADTMO	EINVAL
F_RSP_BAD_TYPE	EINVAL
F_RSP_CBREL_BAD	EIO
F_RSP_CBRES_BAD	EIO
F_RSP_CBNAVL_BAD	EIO
F_RSP_CH_DOWN	EINVAL
F_RSP_CH_INITING	EINVAL
F_RSP_CH_TERMING	EINVAL
F_RSP_CH_UP	EINVAL
F_RSP_CLOSED	EIO
F_RSP_DVRTERM	EIO

F_RSP_FCA1_INFO_BAD	EIO
F_RSP_HALTED	EIO
F_RSP_HALT_FAIL	EIO
F_RSP_IOBMEM	EIO
F_RSP_LMIO_ADR	EIO
F_RSP_LMIO_DIR	EIO
F_RSP_LMIO_HDWR	EIO
F_RSP_LMIO_LEN	EIO
F_RSP_LMIO_ORD	EIO
F_RSP_LOCMEM	EIO
F_RSP_OVERRUN	EIO
F_RSP_PARITY	EIO
F_RSP_PATHCLO	EINVAL
F_RSP_PATHOPN	EINVAL
F_RSP_PKTLEN	EINVAL
F_RSP_RD_PKT_TMO	ELATE
F_RSP_RELMEM	EIO
F_RSP_SECDED	EIO
F_RSP_TERM_FAIL	EIO
F_RSP_TIMER_PAR	EIO
F_RSP_TIMER_QUED	EIO
F_RSP_TIMER_UNKNOWN	EIO
F_RSP_TMIO_ADR	EIO
F_RSP_TMIO_DIR	EIO
F_RSP_TMIO_HDWR	EIO
F_RSP_TMIO_LEN	EIO
F_RSP_TMIO_NAVL	EIO
F_RSP_TMIO_ORD	EIO

EXAMPLES

The following is an example of the usage of the `ioctl` request to alter driver parameters:

```
#include <sys/epackf.h>
#include <sys/fd.h>

struct fdioreq req;                  /* FDDI driver parameter structure */
int ret;                             /* Status returned from ioctl() req */
int fildes;                          /* FDDI device file descriptor */
struct fdio_getset getset;           /* getset structure */

req.buf = &getset;
req.len = sizeof(struct fdio_getset);

ret = ioctl(fildes, FDC_GET, &req);  /* Get current driver settings */
if (ret < 0) {
        perror("FDC_GET");
        exit(1);
};

getset.padcnt = 3;                   /* New padcnt is 3 bytes */
getset.TREQ = 100;                   /* New TREQ is 100 ms */
getset.rtmo = 10;                    /* New read time-out value is 10 sec */

ret = ioctl(fildes, FDC_SET, &cb);   /* Set new driver parameters */
if (ret < 0) {
        perror("FDC_SET");
        exit(1);
};
```

FILES

`/dev/fddi`*n*`/*`	FDDI interface special files
`/usr/include/sys/epackf.h`	
`/usr/include/sys/fd.h`	
`/usr/include/sys/fdsys.h`	
`/usr/include/sys/netdev.h`	

SEE ALSO

`close`(2), `ioctl`(2), `open`(2), `read`(2), `reada`(2), `write`(2), `writea`(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

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

The ANSI documents for FDDI:

FDDI `MAC` (Media Access Protocol) Specification (FDDI-MAC), document number X3.139-1987, November 5, 1986

FDDI `PHY` (Physical Layer Protocol) Specification (FDDI-PHY), document number X3.148-1988, June 30, 1988

FDDI `PMD` (Physical Medium Dependent) Specification (FDDI-PMD), document number X3.166-1990, September 28, 1989

FDDI `SMT` (Station Management) Specification (FDDI-SMT), document number X3T9.5/84-49, Rev 7.2, June 25, 1992

Other documents related to FDDI:

RFC 1390 Transmission of IP and ARP over FDDI Networks. January 1993. D. Katz

Logical Link Control Specification (802.2 LLC), document number 802.2-1985, July 16, 1984

NAME

fei – Front-end interface

IMPLEMENTATION

Cray PVP systems systems

DESCRIPTION

The front-end interface (FEI) is a channel-to-channel adapter that connects Cray PVP systems to a front-end computer. The special file in /dev for the FEI is usually /dev/fei. For details on the special files in /dev at your site, see your system support staff. Currently, support for the FEI is provided through the I/O subsystem (IOS) as if the FEI were a Network Systems Corporation (NSC) adapter, except that the IOS driver provides only one logical connection or logical path. The device is otherwise treated as an NSC adapter; for details of usage, see hy(4).

FILES

/dev/fei

/usr/include/sys/hy.h

/usr/include/sys/hysys.h

SEE ALSO

hy(4), vme(4)

ioctl(2), listio(2), read(2), reada(2), write(2), writea(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

NAME

`fmsg` – GigaRing I/O message and MMR interface

IMPLEMENTATION

CRAY T90 systems with GigaRing-based I/O

CRAY J90 systems with GigaRing-based I/O

DESCRIPTION

The files in `/dev/fmsg` are character special files that allow sending and receiving of message and MMR (Mapped Memory Register) packets to GigaRing I/O nodes. Each file represents one GigaRing I/O node. The following F-transmission protocol packets are supported:

e packets	echo packets
g packets	MMR packets

User-level commands such as `fping`(8) and `mmr`(8) open `/dev/fmsg` devices and send and receive packets using read and write system calls.

The `fping`(8) command sends an echo packet to the specified GigaRing I/O node. The I/O node will then echo the packet back to the sender.

The `mmr`(8) command allows reading and writing of an I/O node's GigaRing MMR for ring management and error monitoring purposes.

The files in `/dev/fmsg` are normally created using the `mkfm`(8) command, based on the `mknod`(8) specifiecation detailed below.

The `mknod`(8) command for `/dev/fmsg` devices is as follows:

`mknod` *name type major minor reserved ionode*

name Name of the `/dev/fmsg` file. Normally named for the I/O node ring and node address.

type Devices in `/dev/fmsg` are character devices denoted by a `c`.

major The major device number is `dev_fmsg`.

minor Minor device number.

reserved Must be 0.

ionode The ring and node address of the target I/O node broken down in octal as follows:

0*rrrnn* where:

rrr = I/O node ring number

nn = I/O node node number

FILES

`/dev/fmsg/*`

`/usr/include/sys/fmsg.h`

`/usr/src/c1/io/fmsg.c`

SEE ALSO

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

NAME

`fslog` – File system error log interface

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `/dev/fslog` pseudo device is a read-only device that holds file system error log records. The file system error log daemon, `fslogd`(8), reads and processes those records to enable graceful handling of file system, directory, and inode errors detected by the kernel. For more information about the file system error log file, see `fslrec`(5).

FILES

`/dev/fslog`	Source of file system error log records

SEE ALSO

`fslrec`(5) for more information about the file system error log file

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

NAME

hdd – HIPPI disk device interface

IMPLEMENTATION

Cray PVP systems with IOS model E

DESCRIPTION

The files in /dev/hdd are special files that allow read and write operations to HIPPI disk array devices. Each file represents one slice of a HIPPI disk device. The files in /dev/hdd are character special files that may be used directly to read and write HIPPI disk slices. Usually, they are called to perform I/O on behalf of higher-level logical disk device drivers. For I/O on a character disk device, read and write operations must transfer multiples of the HIPPI disk device sector size and all seek operations must be on HIPPI disk sector size boundaries.

The UNICOS operating system supports standard HIPPI disk array devices that adhere to the ANSI Standard IPI-3 command set on top of the ANSI Standard HIPPI Framing protocol.

The files in /dev/hdd are not mountable as file systems, although you may combine one or more HIPPI disk slices to make a mountable logical disk device (see dsk(4), ldd(4), and mount(8)). To create the files in /dev/hdd, use the mknod(8) command. Each must have a unique minor device number, along with other parameters used to define a HIPPI disk slice.

The mknod(8) command for HIPPI disk devices is as follows:

mknod *name type major minor dtype iopath start length flags reserved unit ifield*

name Descriptive file name for the device (for example, hdd/scr0230.0).

type Signifies how data will be transferred. Devices in /dev/hdd are character special devices denoted by a c.

major Major device number for HIPPI disk devices. The dev_hdd name label in the /usr/src/uts/c1/cf/devsw.c file denotes the major device number for HIPPI disk devices. You can specify the major number as dev_hdd.

minor Minor device number for this slice. The maximum number of HIPPI disk slices is defined in the deadstart parameter file by the HDDSLMAX parameter.

dtype HIPPI disk device types are defined in /usr/src/uts/c1/sys/pddtypes.h. The HIPPI disk device types currently correspond to the HIPPI disk sector size.

```
HD16    1       /* 16k byte sector, iou = 4  */
HD32    2       /* 32k byte sector, iou = 8  */
HD64    3       /* 64k byte sector, iou = 16 */
```

The iou (I/O unit) is the size of the sector in multiples of 512-word (4096-byte) blocks.

iopath On CRAY Y-MP systems with an IOS-E, the *iopath* specifies the I/O cluster, the I/O processor (IOP), and the controller channel number. For example, an *iopath* of 01234 is IOC 1, IOP 2, channel 34. A HIPPI channel pair takes up two IOP channels, the lower number channel for input and the upper channel number for output. The following is a typical HIPPI IOP configuration that has two pairs of HIPPI channels:

```
HIPPI 0         channel 30      input
                channel 32      output

HIPPI 1         channel 34      input
                channel 36      output
```

When specifying the HIPPI disk *iopath*, the input channel numbers are used (for example, HIPPI 1 on IOC 1, IOP 2, would have an *iopath* of 01234).

For information on setting the *iopath* field when configuring an `hdd` device node on CRAY EL and CRAY J90 systems, see the subsection "The iopath field."

start Absolute starting sector number of the slice.

length Number of blocks sectors in the slice.

flags Flags for HIPPI disk device control, defined in `sys/hdd.h`, follow. They mainly are used for diagnostic and maintenance purposes. Usually, the flags field should be 0 for slices in `/dev/hdd`. The following flags are defined for `hdd` devices.

```
#define HS_CONTROL  0001   /* control device */
#define HS_NODEVINT 0002   /* no device intimate functions */
#define HS_MOVER    0004   /* this dev 3rd party data mover */
#define HS_NOMOVI   0010   /* no intermediate mover response */
#define HS_NOERREC  0040   /* no error recovery */
```

NOTE: If the HIPPI disk array is not a Cray Research supported network disk product, you may have to set the `HS_NODEVINT` flag.

For information on the `HS_DYNPATH` flag, see the subsection "The iopath field."

reserved This field is reserved for future use.

unit This field contains the HIPPI disk array unit number (also known as the *facility address*) and the raid partition number. The low-order 9 bits (bits 0 through 8) represent the facility address, *f*. Bits 9 through 15 represent the raid partition number, *r*: `0`*rrrfff*. To designate an octal value, specify the leading 0 on this parameter in the `mknod` command.

ifield This is the HIPPI array *ifield* address if the array is connected through a HIPPI switch. Bit 2ˆ24 (camp on connect) is forced on by the driver.

The iopath field

On CRAY EL and CRAY J90 systems, you can use the `HS_DYNPATH` flag to create the `hdd` device node by using the `mknod`(8) command.

The `HS_DYNPATH` flag has a value of octal 0400. When the *flags* field in the `hdd` device node has the `HS_DYNPATH` bit set, the *iopath* is treated as a channel mask as opposed to a single channel. When using the channel mask, I/O to or from the `hdd` disk can occur over any of the channels specified in the channel mask.

When the `HS_DYNPATH` bit (octal 0400) is not set in the flags field, I/O to or from the `hdd` disk occurs over the single channel specified in the *iopath* field. There are seven possible input channel values for the *iopath* field for CRAY EL systems: 024, 040, 044, 060, 064, 0100, or 0104. There are 15 possible input channel values for the *iopath* field for CRAY J90 systems: 024, 030, 034, 040, 044, 050, 054, 060, 064, 070, 074, 0100, 0104, 0110, or 0114.

When the `HS_DYNPATH` bit (octal 0400) is set, the *iopath* field in the `hdd` device node represents a bit mask. On CRAY EL systems, this bit mask can be up to 7-bits wide; on CRAY J90 systems, it can be up to 15-bits wide. The bits in the bit mask in CRAY EL systems do not represent the same input channels as the bits in the bit mask in CRAY J90 systems.

The formats of the bit mask in the *iopath* field follow.

The *iopath* bit mask for CRAY Y-MP systems is of the following format (up to 7-bits wide):

Bit	Input channel
2ˆ0	024
2ˆ1	040
2ˆ2	044
2ˆ3	060
2ˆ4	064
2ˆ5	0100
2ˆ6	0104

Examples:

iopath bit mask (binary)	*iopath* bit mask (decimal)	Represents input channels
1000001	65	024, 0104
0111000	56	060, 064, 0100
0000011	3	024, 040
0101010	42	040, 060, 0100

iopath bit mask (binary)	*iopath* bit mask (decimal)	Represents input channels
1010101	85	024, 044, 064, 0104
1110111	119	024, 040, 044, 064, 0100, 0104

The *iopath* bit mask for CRAY J90 systems is of the following format (up to 15-bits wide):

Bit	Input channel
2^0	024
2^1	030
2^2	034
2^3	3040
2^4	044
2^5	050
2^6	054
2^7	060
2^8	064
2^9	070
2^10	074
2^11	0100
2^12	0104
2^13	0110
2^14	0114

Examples:

iopath bit mask (binary)	*iopath* bit mask (decimal)	Represents input channels
000000001000001	65	024, 054
100000000111000	16440	040, 044, 050, 0114
101010101010101	21845	024, 034, 044, 054, 064, 074, 0104, 0114
010101010101010	10922	030, 040, 050, 060, 070, 0100, 0110

Facility addressing

The path to a HIPPI disk facility is defined by the I/O path, *ifield*, and unit number. The `HDDMAX` parameter describes the maximum number of HIPPI disk facilities in the `deadstart` parameter file.

Third-party transfer requests

The IEEE Mass Storage Reference Model breaks the process of doing I/O into its component parts and demonstrates the separation of data and control paths. As looked at from the peripheral's perspective, a control path handles functional request and response information, while the data mover path just moves data. This can allow for centralization of control and can better use high-bandwidth connections for moving and distributing data across a network.

The `hdd` driver has both data mover and data server capabilities. When acting as a data mover, an `hdd` connection is a slave to data transfers being controlled by a server path. When performing the server role, the `hdd` connection sends the read or write function to the device, along with the data path information needed to inform the data mover where to move the data.

To configure an `hdd` connection as a data mover, simply set the `HS_MOVER` flag, as defined previously, in the device inode. When a node is configured as a data mover, the command and response information travel a different path and originate from a server, possibly from a different host. A server functionality is the necessary complement to the data mover function.

Data move functionality for the `hdd` driver is provided with the UNICOS standard `reada`(2) and `writea`(2) system calls. Server functionality is provided through `ioctl` system calls to the `hdd` driver. A unique transfer identifier (tid) logically connects a data move operation with a server request.

`ioctl` requests

The format for `ioctl` requests that the `hdd` driver supports is as follows:

```
#include "sys/pddtypes.h"
ioctl( fildes, command, arg );
```

The `hdd` driver supports the following `ioctl` requests:

`HDI_READFOR` (0101) Reads data to another host.

`HDI_WRITEFOR` (0102) Writes data from another host.

The `HDI_READFOR` and `HDI_WRITEFOR` `ioctl` functions provide the `hdd` driver with a server capability; that is, read and write requests may be sent on behalf of data that is moved down another path to or from another host. You can use the `HDI_READFOR` and `HDI_WRITEFOR` commands in conjunction with an `hdd` node set up to be a data mover. The `hdi_serv` structure passes the appropriate information to the driver.

```
ioctl( fildes, HDI_READFOR, arg )
structure hdi_serve *arg;

/*
 *  Structure for hdd ioctl read/write for
 */
struct hdi_serv {
        uint    nblks   :32,  /* number of 512 word blocks */
                blkno   :32;  /* starting block in 512 word blks */
        uint    tid     :32,  /* transfer identifier */
                offset  :32;  /* byte offset in destination buffer */
        uint    ifield  :32,  /* destination ifield */
                port    :32;  /* destination controller port */
};
```

HDI_CANCEL (0103) Cancels a request to a data mover.

```
int tid;
ioctl( fildes, HDI_CANCEL, tid )
```

The HDI_CANCEL ioctl command cancels a pending data move identified by the transfer identifier (tid).

HDI_GET_TID (0104) Gets a unique transfer identifier (tid).

```
int tid;
ioctl( fildes, HDI_GET_TID, &tid )
```

Gets a system-unique nonzero transfer identifier (tid) that may be used to identify a data mover and/or data server request.

HDI_SET_MOV_TO (0105) Sets data mover request time-out.

```
int timeout = 60;
ioctl( fildes, HDI_SET_MOV_TO, &timeout );
```

Sets the time-out value for a data move to the specified number of seconds. When the timer for a pending data move expires, the pending move is canceled and the data move request is terminated with an EINTR errno.

EXAMPLES

The following mknod command makes a node for hdd/scr0334.2, type c, major number dev_hdd, minor number 110, disk type HD04, I/O cluster 0, IOP 3, channel 34, starting at block 0, length of 100,000 blocks, 0 for flags, facility address of 020, raid partition number 021, and an *ifield* address of 7:

```
mknod hdd/scr0334.2 c dev_hdd 110 1 0334 0 100000 0 0 021020 7
```

FILES

/dev/hdd/*

/usr/include/sys/epackj.h

/usr/include/sys/hdd.h

/usr/include/sys/pddprof.h

/usr/include/sys/pddtypes.h

/usr/src/c1/io/hdd.c

SEE ALSO

dsk(4), ldd(4), mdd(4), pdd(4), xdd(4), sdd(4), ssdd(4)

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

NAME

HIPPI – ANSI High Performance Parallel Interface

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The HIPPI interface (or HIPPI driver) drives an ANSI standard High Performance Parallel Interface (HIPPI) channel. The driver supports no device in particular. If two machines are directly connected by using HIPPI channels, the driver behaves in a manner similar to UNICOS named pipes. If the HIPPI channel is connected to a device, user software must execute the device's protocol. Application processes use the HIPPI driver by means of the standard UNICOS system calls: close(2), ioctl(2), listio(2), open(2), read(2), reada(2), write(2), and writea(2). Each of the special files in a /dev/hippi* directory represents one input or output HIPPI channel.

By convention, HIPPI file names have the following format:

/dev/hippi*n*/i*xx*
/dev/hippi*n*/o*xx*

n Physical channel number

i Input channel

o Output channel

xx Logical channel number

For operation in one direction only, only one channel must be opened. If the application uses both read and write operations, it must open both an input and an output channel. A read operation is valid only on an input channel; a write operation is valid only on an output channel. All HIPPI I/O is raw; the user process is locked in memory as data moves directly between the user buffer and the channel. Therefore, user buffers must be word-aligned, and their length must be a multiple of 8 bytes.

HIPPI Fundamentals

HIPPI is a 32-bit parallel unidirectional point-to-point data channel. Usually, it is installed in input/output pairs for bidirectional operation. Data flows from a HIPPI *source* to a *destination*.

From the user's perspective, the basic unit of information on the channel is a *packet*. The channel hardware breaks up packets into *bursts* of 256 32-bit words. READY pulses from the destination to the source control flow; each READY pulse lets the source send one burst. The PACKET signal from source to destination marks the boundaries between packets.

There is a signal called REQUEST from source to destination, and a companion signal called CONNECT in the opposite direction. When both signals are present, a *connection* is said to exist. Data may not flow unless there is a connection.

To establish a connection, the source raises the `REQUEST` signal to the destination. At this time, it can place 32 bits of information on the data lines, called the *I-field*. The destination can examine the I-field before it responds to the request. The destination responds either by accepting the request or rejecting it. Acceptance consists of raising the `CONNECT` signal and transmitting READY pulses. To reject a request, the destination raises `CONNECT` for a certain period of time and drops it again without transmitting any `READY` pulses. The source responds by dropping `REQUEST`.

Connections may be broken either by the source (by dropping the `REQUEST` signal) or by the destination (by dropping `CONNECT`). The other side must respond by dropping its corresponding signal. The source must cease sending data when the connection is broken. When a destination breaks a connection, data in transit can be lost.

For further information about the `HIPPI` interface, see the ANSI documents listed in the SEE ALSO section.

Dedicated and Shared HIPPI Channels

The `HIPPI` driver supports two modes of operation: dedicated and shared. The mode is assigned to each channel when it is opened. If a channel is dedicated, only one process can have that channel open at a time. If a channel is shared, several processes can use one HIPPI channel. The driver enforces a protocol on the shared channel, allowing it to determine the destination of each incoming message.

Shared channels are not supported on Cray PVP systems configured to read and write the HIPPI channels with SSD solid-state storage buffers.

For more information on configuring HIPPI channels on CRAY Y-MP systems, see *UNICOS Networking Facilities Administrator's Guide,* Cray Research publication SG–2304.

HIPPI `ioctl` Requests

Several `ioctl` requests are available. They have the following format:

```
#include <sys/hx.h>
ioctl (fildes, command, arg)
struct hxio *arg;
```

The valid `ioctl` requests are as follows:

`HXC_GET` Gets the current driver parameter settings; stores them in the `hxio` structure referenced by *arg*.

`HXC_SET` Sets driver parameters from the structure referenced by *arg*.

Connections are established automatically in response to `read`(2) and `write`(2) system calls. To control the termination of connections, use the `HXCF_DISC ioctl` flag (see the following).

The `hxio` structure contains the following fields:

`unsigned int flags` Flags that control channel operation. The *flags* field controls the driver's options. The bits in the flags field are defined as follows:

HXCF_DED — The channel is open in dedicated mode; no other processes may share the channel. If this flag is set, the value in the *path* field is meaningless. The `ioctl` request HXC_GET is used to set this flag. The `ioctl` request HXC_SET ignores this flag.

HXCF_DISC — (Meaningful only on output devices) When set, causes the driver to end the HIPPI connection after each packet written.

HXCF_DOWN — If this bit is set, the channel is unavailable to other users. If this bit is clear, the channel is available. When the channel is down, the driver returns the ENXIO error. Programs that reconfigure the channel should issue a `close` request immediately after the `ioctl` request, then reopen the channel before trying to do anything else. This bit is set in the `ioctl` request HXC_SET on a dedicated channel. This is a read-only flag.

HXCF_HDR — Controls the handling of the HIPPI-FP header (first word of each packet). If HXCF_HDR is set (default), the HIPPI-FP header is assumed to be in the user buffer. If clear, the HIPPI driver adds a HIPPI-FP header to the beginning of the user data on output and strips it off on input. For HIPPI-FP field definitions, see `sys/hippifp.h`. When creating the HIPPI-FP header, the driver sets the `ulpid` field to the value of `path` and the `d2size` field to the user buffer length (in bytes). All other HIPPI-FP fields are 0. The HIPPI driver does not validate the HIPPI-FP header on input.

HXCF_HIPPI — (Read-only, not user settable) Indicates channel is a HIPPI. Clear for HSX.

HXCF_IND — Controls the passing of the I-field value to the driver on output channels. By default, the driver uses the value in `ifv`. If HXCF_IND is set, the driver uses the low-order 32 bits of the first word of the user buffer as the I-field. If HXCF_HDR is set, the HIPPI-FP header is in the second word of the user buffer. HXCF_IND is convenient if the application must change the I-field between packets, especially if `writea`(2) is being used.

HXCF_IO — (Not user-settable) Indicates the channel direction: set for output, and clear for input.

HXCF_ISB — Controls the sending of an Initial Short Burst. The HIPPI-PH specification allows either the first or last burst of a packet to consist of less than 256 HIPPI words (32- or 64-bit words). The HIPPI driver sends short bursts last by default. To send it, the user must first set this flag and use a listio(2) request with two output buffers, and the HXLI_CHD flag set for only the first one. The first buffer contains the contents of the first burst, and it may consist of from 1 to 256 Cray words. The second buffer must be a multiple of 128 Cray words for 32-bit HIPPI channels and 256 Cray words for 64-bit HIPPI channels.

HXCF_MODEL_E — (Not user-settable) Indicates the type of IOS: set for IOS-E and CRAY EL memory HIPPI systems, and clear for others.

`int err` — Detailed error status from the last request; for a list of HIPPI errors, see the Messages section.

`int path` — ULP-ID for shared input channels or for autoheader mode for output channels.

`unsigned int tmo` — Time-out value (in seconds).

`int ifv` — (I-field value) The output channel driver sends this value when it requests a connection. The input channel driver uses this value to decide whether to accept or reject a connection request. The input driver forms the logical product of the channel's I-field value and the user's I-field mask. If this product matches the user's I-field value, the driver accepts the connection. If the values do not match, the driver rejects the connection.

On CRAY EL systems, if the input driver does not use the `ifv` field, all incoming connections are accepted after the input channel has been opened.

`int ifm` — (I-field mask) The input driver makes a logical product with this mask before comparing the I-field on the channel with the I-field value. The `ifm` field is not used on CRAY EL systems.

`int ctmo` — Connection time-out value (in seconds); determines how long the driver will wait for a connection to become established after it is requested.

listio Special Features

The HIPPI driver defines one flag to be used in the `li_drvr` field of the `listreq` structure (see `listio`(2)), as follows:

HXLI_CHD — Indicates, in effect, that user data is chained. When this flag is set, the driver suppresses the end-of-block signal at the end of the current list entry (for an output channel) or allows the current input data block to overflow into the next list entry (for an input channel).

If the driver encounters an end-of-block signal during a read operation in which the HXLI_CHD flag is set, no error indication is returned to the user. Subsequent read operations will complete with a data length of 0 until the HXLI_CHD flag is cleared. If the user issues the ioctl request HXC_GET to check the status after a chained read operation, the err field of the hxio structure is set to the HXST_EOB error code, indicating that an end-of-block signal arrived before the last read request was processed.

If the driver has not detected an end-of-block signal on a unchained read operation (that is, read(2), reada(2), or listio(2) with the HXLI_CHD flag clear) by the time the read operation completes, the driver discards the unread part of the block. The byte count returned to the user is equal to the size of the buffer, and no error code is returned in errno. If the user issues the ioctl request HXC_GET to check the status after a unchained read operation, the err field of the hxio structure is set to the HXST_LONG error code, indicating that the entire block was not read and the remainder was discarded.

When using the listio chaining feature on shared channels to combine more than one buffer to make a single data block, the user must include all buffers in the same listio(2) system call. That is, make sure that the HXLI_CHD flag is clear in the li_drvr field of the last item for an HIPPI channel in each listio(2) call. Failure to do this can cause lost or corrupt data on the channel if the user process is swapped between requests.

The IOS-E systems support chained buffers only in pairs; that is, if HXLI_CHD is set for one buffer, it must be clear for the next.

On CRAY EL systems with memory HIPPI, the chained buffers also must be supplied in pairs. The first buffer also must consist of 1024 bytes or less. On the input channel, the first burst will be written to the first buffer. If the first burst fits completely into the first buffer, the remainder of the packet will be written to the second buffer, leaving empty space in the first buffer if the first burst did not fill it completely. If the first burst is bigger than the first buffer, the first buffer will be filled, the remainder of the first burst will be written to the second buffer, and the remainder of the packet will be appended to the second buffer.

HIPPI Protocol

On dedicated channels, the HIPPI driver treats all HIPPI packets as opaque data: no specific protocol is required or recognized.

Shared channel operation requires that all applications use the ANSI draft HIPPI Framing Protocol (HIPPI-FP). The sys/hippifp.h header file contains definitions for the header. The HIPPI driver examines the ULP-ID field in each input packet. For an application to receive a packet in shared mode, its *path* value must match the contents of this field. The default value of *path* is *n*+128; *n* = (minor device number) modulo 16 (for example, the default path (ULP-ID) for /dev/hippi/i01 and /dev/hippi/o01 is 129; for /dev/hippi/*02 is 130, and so on). Each application can change its ULP-ID by setting the new value in the *path* variable in a HXC_SET request.

HIPPI 800- and 1600-M Modes

The HIPPI-PH specification describes both 32-bit and 64-bit HIPPI implementations. Nominal data rates are 800 M and 1600 M, respectively. One cable in each direction, called Cable A, is used for 800-M HIPPI. 1600-M HIPPI uses two cables in each direction, called Cable A and Cable B. Cray Research systems that have an IOS-E can be wired for Cable B; all other Cray Research systems have only 32-bit HIPPI.

The ANSI HIPPI-SC (Switch Control) draft standard allocates bit 2**28 of the I-field to control the mode of switch connections. The Cray Research HIPPI driver uses this same bit to select the channel mode, as follows:

Output If bit 2**28 of the I-field is set and the output HIPPI has a second cable installed and connected, the driver selects 64-bit mode for the transfer. If there is no second cable, the driver returns EIO.

Input The driver examines bit 2**28 of incoming I-fields. If set, and Cable B is installed and connected, the driver selects 64-bit mode for the transfer.

MESSAGES

When a fatal error occurs, the HIPPI driver returns one of the following error codes in errno. The error codes and their meanings are as follows:

EFAULT A bad argument address was specified in an ioctl request.

EINVAL The driver software has detected a fatal parameter error. Errors in system configuration and user errors in system call invocation, can cause this error.

EIO One of the following conditions can cause this error.

- A fatal I/O error occurred or the HIPPI channel closed while asynchronous I/O was active (on a read(2) or write(2) system call).
- A data block was too long for the input buffer (on a read(2) system call) or overflowed the channel (on a write(2) system call).
- An I/O request timed out (on a read(2) or write(2) system call).

The detailed error status is available in the err field of the hxio structure; to see this field, use the ioctl request HXC_GET.

ELATE An input request timed out.

ENXIO The HIPPI channel is unavailable (on an open(2) system call), or the channel is not open (on a close(2) system call). On Cray PVP systems, this code can mean that the IOP failed to allocate some resource.

After a fatal error, the detailed error status is available with the ioctl request HXC_GET. The driver returns error codes in the err field of the hxio structure.

The following detailed error codes are defined on CRAY EL systems that have VME HIPPI:

HXST_BUF IOS I/O buffer unavailable on open operation.

HXST_CHAN CPU gave bad channel number to IOS; caused by configuration error.

HXST_DBG Debug mode error; indicates a driver fault (should never occur).

HXST_EOB Unexpected end of packet (input only).

HXST_FLGS Buffer flags do not match; indicates a driver fault (should never occur).

HXST_FMEM	IOS free memory unavailable on open operation.
HXST_FNC	Illegal function code; indicates a driver fault (should never occur).
HXST_HISP	No high-speed channel to this IOP; caused by configuration error or wrong target memory.
HXST_LLEN	Transfer length too long; indicates a driver fault (should never occur).
HXST_LONG	Long block received (input only).
HXST_MOS	MOS buffer unavailable on open operation (debug mode only).
HXST_NDEV	No device present on the channel (hardware signal).
HXST_OK	No error (binary 0).
HXST_OPEN	Channel is not open; indicates a driver fault (should never occur).
HXST_OVER	Data overrun error (input only).
HXST_TM	Bad target memory type; indicates a driver fault (should never occur).
HXST_TMO	Request timed out.
HXST_ZLEN	Buffer length is 0; indicates a driver fault (should never occur).
HXST_CTMO	Connection timed out.
HXST_CNPR	Connection not present.
HXST_CREJ	Connection rejected.
HXST_DISC	Channel disconnected.
HXST_CNPE	No connection pending.
HXST_CAPR	Connection already present.
HXST_CABT	Connection aborted.
HXST_LLRC	Length/Longitudinal Redundancy Checkword (LLRC) error.
HXST_CPE	Channel parity error.
HXST_CHST	Channel status is not valid.
HXST_BOE	Buffer overrun error.
HXST_OIM	Odd initial microburst.
HXST_BPE	Buffer parity error.
HXST_IPE	I-field parity error.

The following detailed error codes are defined for Cray Research systems that have an IOS-E and CRAY EL systems that have VME HIPPI:

HIST_INV_REQ	Request code is not valid.
HIST_INV_RL	Request packet length is not valid.

HIST_IS_UP	Channel already configured up.
HIST_NOT_UP	Channel is not configured up.
HIST_NOT_IMP	Function is not implemented.
HIST_RTMO	Read request time-out.
HIST_DRV_DOWN	Driver terminated by configuration down.
HIST_IS_OPN	Logical path already open.
HIST_RAW_OPN	Bad raw channel open request.
HIST_NOT_OPN	Logical path is not open.
HIST_PTH_CLS	Read abandoned because path is closed.
HIST_LONG	Long packet excess discarded (nonfatal).
HIST_DTA_ERR	Data integrity error on read.
HIST_CHAN_TMO	Channel activation time-out.
HIST_DRV_TERM	Driver terminating.
HIST_NOT_CNCT	Interconnect-A not present.
HIST_HALT_IO	Read/write request returned due to halt-io.
HIST_INV_PARM	Configure UP parameter that is not valid.
HIST_NOT_64	Cannot write in 64-bit HIPPI mode.
HIST_CN_REJ	Connection attempt was rejected.
HIST_CN_FAIL	Connection attempt failed (other).
HIST_CN_TMO	Connection request time-out.
HIST_CN_STUCK	Connect-in will not drop.
HIST_INV_SF	Device control subfunction that is not valid.
HIST_HANGUP	Connection went away.
HIST_SECDED	SECDED error in I/O buffer.
HIST_INTRCNCT	Lost Interconnect.
HIST_EOB	Short block received.

The following detailed error codes are defined for IOS-E systems:

HIST_RC_OK	No errors
HIST_RC_PARAM	Parameter error
HIST_RC_NO_EVENT	Requested event not outstanding
HIST_RC_NOT_QUED	Entry was not on specified queue

`HIST_RC_Q_EMPTY`	Queue is empty
`HIST_RC_BAD_CB`	Invalid I/O buffer ordinal
`HIST_RC_BAD_ADR`	Invalid I/O buffer address
`HIST_RC_INVDIR`	Invalid transfer direction
`HIST_RC_TMNAVL`	Target memory channel not available/configured
`HIST_RC_HWERR`	Unrecovered hardware error
`HIST_RC_QUEUED`	Entry already on a queue
`HIST_RC_NOMEM`	Memory space not available
`HIST_RC_BADLEN`	Bad memory allocation length
`HIST_RC_BADREQPKT`	Bad request packet address on release
`HIST_RC_BADRESPKT`	Bad respond packet address
`HIST_RC_QUEREQ`	Must queue request (can't set pointer)
`HIST_RC_CBNAVL`	Channel buffer not available to reserve
`HIST_RC_CBNOTOWN`	Channel buffer not owned by subsystem
`HIST_RC_BADSSID`	Invalid subsystem identifier

EXAMPLES

The following is an example of the usage of the `hxio` structure to alter driver defaults on CRAY Y-MP systems:

```
#include <sys/hx.h>
struct hxio cb; /* HIPPI driver parameter structure              */
int s;          /* Status returned from ioctl() request          */
int fildes;     /* HIPPI output device file descriptor           */

s = ioctl(fildes, HXC_GET, &cb);  /* Get current driver settings  */
if(s  < 0)      {
        perror("HXC_GET");
        exit(1);
}
cb.flags |= HXCF_DISC;  /* Automatic disconnect after each output */
cb.ifv = ifield;        /* I-field value for all packets          */
         /* OR */
cb.flags |= HXCF_IND;   /* I-field prefixed to data in buffer     */
cb.path = 0300;         /* Set new ULP-ID value                   */
cb.ctmo = 10;           /* Connection timeout value is 10 seconds */
s = ioctl(fildes, HXC_SET, &cb);  /* Set new driver parameters    */
if(s  < 0)      {
        perror("HXC_SET");
        exit(1);
}
```

FILES

/dev/hippi*/*	HIPPI channel special files
/usr/include/sys/hx.h	
/usr/include/sys/hpacket.h	(CRAY Y-MP systems and CRAY EL systems with VME)
/usr/include/sys/hxsys.h	(CRAY Y-MP systems and CRAY EL systems with VME)

SEE ALSO

`hsx`(4)

`close`(2), `ioctl`(2), `listio`(2), `read`(2), `reada`(2), `write`(2), `writea`(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

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

General UNICOS System Administration, Cray Research publication SG–2301

The ANSI documents for `HIPPI`:

HIPPI Mechanical, Electrical, and Signaling Protocol Specification (HIPPI-PH), document number X3T9.3/88-127, Rev 8.1, June 24, 1991

HIPPI Framing Protocol (HIPPI-FP), document number x3T9.3/89-146, Rev 4.2, June 24, 1991

HIPPI 802.2 Link Encapsulation (HIPPI-LE), document number X3T9.3/90-119, Rev 3.1, June 28, 1991

HIPPI Physical Switch Control (HIPPI-SC), document number X3T9.3/91-023, Rev 1.9, June 28, 1991

NAME

`hpi3` – IPI-3/HIPPI packet driver configuration file

IMPLEMENTATION

Cray PVP systems that have an IOS model E

DESCRIPTION

The IPI-3/HIPPI packet driver configuration file consists of statements that describe the I/O processors (IOPs), channels, slaves, and devices that compose the IPI-3/HIPPI subsystem. It also includes a list of driver options and limits that, when specified, override the system defaults.

This man page describes the format of the configuration file and the IPI-3/HIPPI packet driver `ioctl` requests. For information on the IPI-3/HIPPI packet driver commands, see the `hpi3_clear`(8), `hpi3_config`(8), `hpi3_option`(8), `hpi3_start`(8), `hpi3_stat`(8), and `hpi3_stop`(8) man pages in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022.

Configuration File Format

Each line in the configuration file is a configuration type statement, a configuration description statement, a comment, or white space.

To specify comments, begin a line with the # symbol. Configuration type statements begin with the `-` symbol, followed by one of these strings: `IOPS`, `CHANNELS`, `SLAVES`, `DEVICES`, or `OPTIONS`. You must specify the configuration type statements in the order listed. All lines specified between configuration type statements that are not comments or white space are configuration description statements. The format of the configuration description statements depends on the configuration type statement preceding it.

The configuration file format is as follows:

`-IOPS`
iop description statements (CRAY Y-MP systems with IOS model E (IOS-E) only)

`-CHANNELS`
channel description statements

`-SLAVES`
slave description statements

`-DEVICES`
device description statements

`-OPTIONS`
option description statements

IOP Description

(CRAY Y-MP systems with an IOS-E only) The IOP description statements follow the `-IOPS` statement. Each IOP description statement describes one IOP. You can specify 32 IOP description statements. You cannot specify an IOP description for CRAY EL series and CRAY J90 series systems. The description statement must be in the following format:

iop-name cluster-number iop-number

iop-name — Consists of 1 to 8 characters and must be unique. The name is used in creating a file that can be used to issue packets that affect the IOP as a whole, rather than device-specific requests.

cluster-number — Must be in the range 0 to 15.

iop-number — Must be in the range 0 to 3. Each cluster-IOP pair must be unique.

Channel Description

The channel description statements follow the `-CHANNELS` statement. Each channel statement describes one channel. The description statement must be in the following format for CRAY Y-MP system with IOS-E:

iop-name input-channel output-channel state input-ch-timeout output-ch-timeout connect-timeout

The description statement must be in the following format for CRAY J90 and CRAY EL systems:

input-channel output-channel state input-ch-timeout output-ch-timeout connect-timeout

iop-name — (CRAY Y-MP systems with an IOS-E only) Denotes the name of the IOP in which the channel is attached. The IOP must already be defined in one of the IOP description statements.

input-channel — Specifies the channel number of the input subchannel. The subchannel is attached to EIOP *iop-name* on CRAY Y-MP systems with an IOS-E. For a list of valid channel pairs for CRAY J90 and CRAY EL systems, see the Channel Selection (CRAY J90 and CRAY EL systems) subsection.

output-channel — Specifies the channel number of the output subchannel. On CRAY Y-MP systems with an IOS-E, the subchannel is attached to EIOP *iop-name*, the channel numbers must be 030, 032, 034, or 036, and a channel number must be unique within an EIOP. For a list of valid channel pairs for CRAY J90 and CRAY EL systems, see the Channel Selection (CRAY J90 and CRAY EL systems) subsection.

state — Specifies the status (`UP` or `DOWN`) of the channel after starting the packet driver. If you specify `UP`, the channel pair will be configured up as part of the start-up sequence. If you specify `DOWN`, the channel pair is left down after the start-up sequence completes.

input-ch-timeout — Specifies the time-out period (in milliseconds) for requests issued to the input channel. The time-out period must be in the range 0 to 0177777. If you specify 0, the default time-out period is 10 seconds.

output-ch-timeout Specifies the time-out period (in milliseconds) for requests issued to the output channel. The time-out period must be in the range 0 to 0177777. If you specify 0, the default time-out period is 10 seconds.

connect-timeout Specifies the time-out period (in hundredths of a second) for the connection request. The time-out period and must be in the range 0 to 0177777. If you specify 0, the default time-out period is 10 seconds.

On CRAY Y-MP systems with an IOS -E, you can specify a maximum of two channel descriptor statements per IOP. CRAY EL systems can have up to 7 memory-HIPPI channels, and CRAY J90 systems can have up to 15 memory-HIPPI channels; for a list of valid channel pairs for CRAY J90 and CRAY EL systems, see the Channel Selection (CRAY J90 and CRAY EL systems) subsection.

Slave Description

The slave description statements follow the `-SLAVES` statement. Each slave statement describes one slave. The description statement must be in the following format for CRAY Y-MP systems with an IOS-E:

slave-name iop-name channel-pairs(s) i-field

The description statement must be in the following format for CRAY J90 and CRAY EL systems:

slave-name channel-pairs(s) i-field

slave-name Denotes the name of the slave (attached to IOP *iop-name* on CRAY Y-MP systems with an IOS-E). A slave name must consist of 1 to 8 characters and must be unique. The slave name is used in the device definition to identify the paths to the device.

iop-name (CRAY Y-MP systems with an IOS-E) Denotes the name of the IOP in which the slave is attached. The IOP must already be defined in one of the IOP description statements.

channel-pair(s) Specifies the channel pairs that may be used to issue requests to the slave and to transfer data to the slave. The channel pairs must be in the following format:

input-channel1:output-channel1[`,` *input-channel2:output-channel2*]

If you specify two channel pairs, the channel pairs must be unique. The channel pairs specified must already be defined in a channel description statement. See the Channel Selection (CRAY J90 and CRAY EL systems) subsection for a list of valid channel pairs for CRAY J90 and CRAY EL systems.

i-field The HIPPI i-field address. It is used to route requests and data from a HIPPI switch to a slave.

On CRAY Y-MP systems with an IOS-E, you may specify a maximum of 32 slaves per IOP.

Device Description

The device description statements follow the `-DEVICES` statement. Each device statement describes one device. The description statement must be in the following format:

device-name slave-name low-facility-address high-facility-address

device-name Specifies the name of a device attached to slave *slave-name*. The device name must consist of 1 to 8 characters and must be unique. The name is used to create a file that can be used to issue packets to the device.

slave-name Denotes the name of the slave in which the device is attached. The slave must have been defined in one of the slave description statements.

Facility addresses are used to route requests or data from a slave to a device. *low-facility-address* and *high-facility-address* specify a range of facility addresses. Asynchronous responses with facility addresses within this range are associated with the device. The facility addresses specified must be in the range 0 to 0xFF.

You may specify a maximum of 32 devices.

Option Description

The option description statements follow the `-OPTION` statement. Each option statement is in the format:

option option-value

Valid options are `IOS_OPTIONS`, `MAX_ASYNC`, `MAX_IOP_PROC`, `MAX_NON_CMDLST`, `MAX_STK_COUNT`, and `TRACING`.

`IOS_OPTIONS` Defines a value used to control temporary and installation-specific IOP configuration options. If you omit this option, the IOS option value defaults to 0.

`MAX_ASYNC` Defines the maximum number of asynchronous responses that may be enabled for an individual device. The number of asynchronous responses must be in the range 0 to 20. If you omit this option, the packet driver will default to a maximum of five asynchronous responses.

`MAX_IOP_PROC` Defines the maximum number of processes that can open an IOP device concurrently. The number of processes must be in the range 1 to 50. If you omit this option, the packet driver will default to a maximum of 10.

`MAX_NON_CMDLST` Defines the maximum number of noncommand list requests that may be stacked for an individual device. The stack count must be in the range 0 to 10. If you omit this option, the packet driver will default to a maximum of five.

`MAX_STK_COUNT` Defines the maximum number of command list requests that may be stacked for an individual device. The stack count must be in the range 0 to 20. If you omit this option, the packet driver will default to a maximum of five.

`TRACING` Specifies whether packet driver tracing should be on or off. The option value must be either `ON` or `OFF`.

Channel Selection (CRAY J90 and CRAY EL systems)

The following table indicates valid channel pairs for CRAY J90 systems:

Channel pair	Input channel	Output channel
1	024	027
2	030	033
3	034	037
4	040	043
5	044	047
6	050	053
7	054	057
8	060	063
9	064	067
10	070	073
11	074	077
12	0100	0103
13	0104	0107
14	0110	0113
15	0114	0117

The following table indicates valid channel pairs for CRAY EL systems:

Channel pair	Input channel	Output channel
1	024	027
2	040	043
3	044	047
4	060	063
5	064	067
6	0100	0103
7	0104	0107

Sample Configuration Files

The following is an example of an IPI-3/HIPPI packet driver configuration file.

```
#
#   IPI-3/HIPPI Packet Driver Configuration File
#

#
#   Define the IOPs.
#

-IOPS

#
#IOP Name     Cluster   IOP
#
iop_0_0          0       0
iop_0_2          0       2
iop_3_2          3       2
iop_3_3          3       3

#
#   Define the channels
#

-CHANNELS

#
#IOP Name
#      +Input Channel
#              +Output Channel
#                          +State
#                           (UP,DOWN)
#                                  +Input Channel
#                                   Timeout
#                                   (sec/1000)
#                                             +Output Channel
#                                              Timeout
#                                              (sec/1000)
#                                                         +Connection
#                                                          Timeout
#                                                          (sec/1000)

iop_0_0  030     032         UP      3000       3000         2000
iop_0_0  034     036         UP      3000       3000         2000
iop_0_2  030     032         DOWN    1000       1000         2000
iop_3_2  034     036         UP      2000       2000         3000
iop_3_3  034     036         DOWN    2000       3000         2000
```

```
iop_3_3  030     032          UP      3000        2000         3000

#
#   Define the slaves
#

-SLAVES

#
#Slave Name   IOP NAME    Channel Pair(s)   I-Field
#

slv_0         iop_0_0     030:032/034:036   00x01000007

slv_1         iop_0_2     030:032           00x01000007
slv_2         iop_0_2     030:032           00x01000008
slv_3         iop_0_2     030:032           00x01000009
slv_4         iop_0_2     030:032           00x0100000a

slv_5         iop_3_2     030:036           00x01000003

slv_6         iop_3_3     030:032           00x01000004
slv_7         iop_3_3     030:032           00x01000005
slv_8         iop_3_3     030:036           00x01000007

#
#   Define the devices
#

-DEVICES

#Device    Slave      Low Facility  High Facility
#Name                 Address       Address
#

er91       slv_0      0x01          0x01
er92       slv_0      0x02          0x02
er93       slv_0      0x03          0x03
er94       slv_0      0x04          0x04

dsk_1      slv_1      0xFF          0xFF
dsk_2      slv_2      0xFF          0xFF
dsk_3      slv_3      0xFF          0xFF
dsk_4      slv_4      0xFF          0xFF

hpdsk      slv_5      0xFF          0xFF

d1_1       slv_6      0xFF          0xFF
```

```
d1_2       slv_7    0xFF        0xFF
hpdsk_2    slv_8    0xFF        0xFF
```

The following is an example of a configuration file for a CRAY EL system:

```
################################################################################
#                                                                              #
#     Tape K-packet driver configuration file for CRAYELS systems              #
#                                                                              #
################################################################################
#
#    IPI-3 Tape Configuration
#

-CHANNELS

# +Input Channel
# |   +Output Channel
# |   |
# |   |     +State (UP,DOWN)
# |   |     |     +Input Channel
# |   |     |     |  timeout  +Output Channel
# |   |     |     |  (sec/1000)     |  timeout  +Connection
# |   |     |     |           |  (sec/1000)     |  timeout
# v   v     v     v           v           v  (sec/100)

0104  0107  UP    30000       30000       2000


-SLAVES
#
# +Tape Slave Name
# |         +Channel Pair(s)
# |         |
# |         |                             +I-Field
# |         |                             |
# v         v                             v

s_hippi1    0104:0107                     0x0100001d


-DEVICES
#
# +Tape Device Name (for /dev/hpi3 )
# |              +Slave name
# |              |           +Low Facility Address
# |              |           |     +High Facility Address
```

```
# |              |            |     |
# |              |            |     |
# |            |        |     |
# |              |            |     |
# v              v            v     v

hdd_dsk1     s_hippi1   0xFF  0xFF


-OPTIONS

#   Define the maximum number of command list requests
#   that may be stacked for an individual device

MAX_STK_COUNT           5


#   Define the maximum number of non-command list requests
#   that may be stacked for an individual device

MAX_NON_CMDLST           3


#   Define the maximum number of asynchronous responses
#   that may be enabled for an individual device

MAX_ASYNC         5


#   Define the maximum number of processes that can
#   open an iop device

MAX_IOP_PROC            10


#   Request kernel tracing to be on or off.

TRACING           On


#   IOS_OPTIONS defines a value used to control
#   temporary and installation-specific configuration options.

# IOS_OPTIONS     0xabc
```

ioctl Requests

You can use the following ioctl(2) requests with the IPI-3/HIPPI packet driver: PKI_DRIVER_STS, PKI_ENABLE, PKI_GET_CONFIG, PKI_GET_DEVCONF, PKI_GET_DEVTBL, PKI_GET_OPTIONS, PKI_RECEIVE, PKI_SEND, and PKI_SIGNO. A description of each of these requests follows, along with an example of its use.

PKI_DRIVER_STS

The `PKI_DRIVER_STS` request returns the status of the IPI-3/HIPPI packet driver. If the packet driver has been started, a negative value is returned. If the packet driver is down, 0 is returned.

You can issue this request to the request device, IOP devices, or IPI-3 devices.

The following example shows how to issue a request for the driver status:

```
/*
 *    Get the IPI-3/HIPPI Packet Driver Status
 */

#include <sys/types.h>
#include <sys/fcntl.h>
#include <errno.h>
#include <sys/cnthpi3.h>
#include <sys/pki_ctl.h>

main()
{
           int         fd;

           /*
            *     Open the request device
            */

           if ( (fd = open(HPI3_REQ, O_RDWR ) ) < 0 ) {
              perror( "Unable to open the request device" );
              exit(errno);
           }

           if ( ioctl( fd, PKI_DRIVER_STS, 0 ) < 0 ) {
                    printf( "The packet driver has been started");
           } else {
                    printf( "The packet driver is not active" );
           }

           close(fd);
}
```

PKI_ENABLE

The `PKI_ENABLE` request enables the packet interface for a device. The driver allocates buffer space for the packets issued to the driver. You must issue this request before trying to register for a signal and before trying to send request packets.

You can issue this request only to IOP devices and IPI-3 devices.

The following example shows how to issue a request to enable the packet interface:

```
/*
 *    Enable the packet interface
 */

#include <sys/types.h>
#include <sys/fcntl.h>
#include <errno.h>
#include <sys/cnthpi3.h>
#include <sys/pki_ctl.h>

main()
{
            int         fd;

            /*     Open the  device  */

            if  (  (fd = open("/dev/ipi3/iop_3", O_RDWR )  )  < 0  )  {
                    perror( "Unable to open the device" );
                    exit(errno);
            }

            /*   Enable the packet interface   */

            if  ( ioctl( fd, PKI_ENABLE, 0 )  <  0  )  {
                     perror( "Unable to enable the packet interface");
            }

            close(fd);
}
```

If no error has occurred, the `ioctl` return code will be 0. If an error has occurred, the error code will be one of the following codes (in decimal):

`EPKI_IOCTL_REQT`	415	`ioctl` is not valid for the device type.
`EPKI_ALREADY_ENBL`	423	The packet interface has already been enabled.

The symbols for these error codes are located in file `errno.h`.

PKI_GET_CONFIG

The `PKI_GET_CONFIG` request returns the IPI-3/HIPPI packet driver configuration. The configuration is returned in structure `cnthpi3`, which is defined in the `sys/cnthpi3.h` file.

The argument to the `ioctl` system call is a pointer to structure `pki_ctl`, described in the `sys/pki_ctl.h` file. The `pki_packet` field of structure `pki_ctl` must be set to a pointer to structure `cnthpi3`.

You can issue this `ioctl` only to the request device.

The following example shows how to issue a request for the configuration:

```
/*
 *    Get the IPI-3/HIPPI configuration.
 */

#include <sys/types.h>
#include <sys/fcntl.h>
#include <errno.h>
#include <sys/cnthpi3.h>
#include <sys/pki_ctl.h>

main()
{
            struct  pki_ctl    ctl;
            struct  cnthpi3  cnt;

            int         fd;

            /*
             *     Open the request device.
             */
            if  (   ( fd = open(HPI3_REQ, O_RDWR )  )  <  0  ) {
              perror( "Unable to open the request device" );
              exit(errno);
            }

            ctl.pki_packet = (word *)&cnt;

            if  (  ioctl( fd, PKI_GET_CONFIG, &ctl )  <  0  ) {
                    perror( "Unable to get the configuration");
            }

            close(fd);
}
```

If no error has occurred, the `ioctl` return code will be 0. If an error has occurred, the error code will be one of the following codes (in decimal):

`EFAULT`	14	An address specified points outside the user's address space.
`EPKI_IOCTL_REQT`	415	`ioctl` is not valid for the device type.

The symbols for these error codes are located in file `errno.h`.

PKI_GET_DEVCONF

The `PKI_GET_DEVCONF` request returns the configuration of an IPI-3/HIPPI device. The configuration is returned in structure `cnthpi3_entry`, which is defined in the `sys/cnthpi3.h` file.

The argument to the `ioctl` system call is a pointer to structure `pki_ctl`, described in the `sys/pki_ctl.h` file. The `pki_packet` field of structure `pki_ctl` must be set to a pointer to structure `cnthpi3_entry`. The `pki_device` field of structure `pki_ctl` must be set to the name of the device.

You can issue this `ioctl` only to the request device.

The following example shows how to issue a request for the device configuration:

```
/*
 *    Get the IPI-3/HIPPI device configuration for device, d1dev
 */

#include <sys/types.h>
#include <sys/fcntl.h>
#include <errno.h>
#include <sys/cnthpi3.h>
#include <sys/pki_ctl.h>

main()
{
            struct  pki_ctl              ctl;
            struct  cnthpi3_entry  ce;

            int         fd;

            /*
             *      Open the request device
             */
            if  (   ( fd = open(HPI3_REQ, O_RDWR )  )  <  0  )  {
              perror( "Unable to open the request device" );
              exit(errno);
            }

            ctl.pki_packet  = (word *)&ce
            ctl.pki_device  = 0
            strncpy( (char *)&ctl.pki_device, "d1dev", strlen("d1dev") );

            if  (  ioctl( fd, PKI_GET_DEVCONF, &ctl )  <  0 ) {
                     perror( "Unable to get the device configuration");
            }

            close(fd);
```

```
}
```

If no error has occurred, the `ioctl` return code will be 0. If an error has occurred, the error code will be one of the following codes (in decimal).

`EFAULT`	14	An address specified points outside the user's address space.
`EPKI_IOCTL_REQT`	415	`ioctl` is not valid for the device type.
`EPKI_NO_DEVICE`	421	The device specified is not in the configuration.

The symbols for these error codes are located in file `errno.h`.

PKI_GET_DEVTBL

The PKI_GET_DEVTBL request returns the device table of an IPI-3/HIPPI device. The device table is returned in structure hpi3_tab, which is defined in the sys/hpi3.h file.

The argument to the ioctl system call is a pointer to structure pki_ctl, described in the esys/pki_ctll.h file. The pki_packet field of structure pki_ctl must be set to a pointer to structure hpi3_tab. The pki_device field of structure pki_ctl must be set to the name of the device.

You can issue this ioctl only to the request device.

The following example shows how to issue a request for the device table:

```
/*
 *    Get the IPI-3/HIPPI device table for device, d1dev
 */

#include <sys/types.h>
#include <sys/fcntl.h>
#include <errno.h>
#include <sys/epack.h>
#include <sys/hpi3.h>
#include <sys/cnthpi3.h>
#include <sys/pki_ctl.h>

main()
{
            struct  pki_ctl      ctl;
            struct  hpi3_tab  tab;

            int         fd;

            /*
             *     Open the request device
             */
            if  (  ( fd = open(HPI3_REQ, O_RDWR )  )  <  0  )  {
              perror( "Unable to open the request device" );
              exit(errno);
            }

            ctl.pki_packet  = (word *)&tab
            ctl.pki_device  = 0
            strncpy( (char *)&ctl.pki_device, "d1dev", strlen("d1dev") );

            if  (  ioctl( fd, PKI_GET_DEVTBL, &ctl )  <  0 ) {
                     perror( "Unable to get the device table");
```

```
        }

        close(fd);
}
```

If no error has occurred, the ioctl return code will be 0. If an error has occurred, the error code will be one of the following codes (in decimal):

EFAULT	14	An address specified points outside the user's address space.
EPKI_IOCTL_REQT	415	ioctl is not valid for the device type.
EPKI_NO_DEVICE	421	The device specified is not in the configuration.

The symbols for these error codes are located in file errno.h.

PKI_GET_OPTIONS

The PKI_GET_OPTIONS ioctl request returns the IPI-3/HIPPI packet driver options. The options are returned in structure pki_option, which is defined in the sys/pki_ctl.h file. Five options are returned from the driver.

Field pki_max_async defines the maximum number of asynchronous responses that may be enabled for an individual device. Field pki_max_list_cmd defines the maximum number of command list requests that may be stacked for an individual device. Field pki_max_iop_proc defines the maximum number of processes that can open an IOP device concurrently. Field pki_max_non_cmdlst defines the maximum number of noncommand list requests that may be stacked for a device. If packet driver tracing is on, field pki_tracing is set to 1; if it is set to 0, it is off.

To display the options, use the IPI-3/HIPPI command hpi3_stat(8) with option -o:

```
hpi3_stat -o
```

The following example shows how to issue a request for the options:

```
/*
 *    Get the IPI-3/HIPPI options
 */

#include <sys/types.h>
#include <sys/fcntl.h>
#include <errno.h>
#include <sys/pki_ctl.h>
#include <sys/cnthpi3.h>

main()
{
            struct  pki_option    option;

            int         fd;

            /*
             *     Open the request device
             */
            if  (  ( fd = open(HPI3_REQ, O_RDWR )  )  <  0  )  {
              perror( "Unable to open the request device" );
              exit(errno);
            }

            if  (  ioctl( fd, PKI_GET_OPTIONS, &option )  <  0 ) {
                     perror( "Unable to get the options");
            }

            close(fd);
}
```

If no error has occurred, the `ioctl` return code will be 0. If an error has occurred, the error code will be one of the following codes (in decimal):

`EFAULT`	14	An address specified points outside the user's address space.
`EPKI_IOCTL_REQT`	415	`ioctl` is not valid for the device type.

The symbols for these error codes are located in file `errno.h`.

PKI_RECEIVE

The `PKI_RECEIVE` `ioctl` request is used to acquire the next response packet from the IPI-3/HIPPI packet driver.

The argument to the `ioctl` call is a pointer to structure `pki_ctl`, which is defined in the `sys/pki_ctl.h` file. The `pki_packet` field of structure `pki_ctl` must be set to a pointer to a buffer large enough to receive the IPI-3/HIPPI IOP response. The `pki_nbytes` field of structure `pki_ctl` must be set to the size of the buffer.

A packet will be returned only if the following conditions are met:

- The packet interface must be enabled
- A response packet must be available
- The buffer space must be large enough to accommodate the packet

You can issue this `ioctl` only to IOP devices and IPI-3 devices.

You may use the C library routine `ipi_get_pkt` to acquire packets. This routine creates and then issues a `PKI_RECEIVE ioctl` request until a packet is obtained or an error other than `EPKI_NO_PACKETS` is received. When a response packet is not available, `EPKI_NO_PACKETS` is the error code returned.

The following example shows how to receive an IOP packet:

```
/*
 *   Receive an IOP packet
 */

#include <sys/types.h>
#include <sys/fcntl.h>
#include <errno.h>
#include <sys/epack.h>
#include <sys/epackk.h>
#include <sys/cnthpi3.h>
#include <sys/pki_ctl.h>

main()
{
            struct  pki_ctl    ctl;

            char pkt[HPI3_PKT_SIZE];

            int         reply;
            int         fd;

            /*
             *    Open the  IPI-3 device
             *      ipi_open is a library routine that registers for signal
             *      IPI_PKT_SIG, opens the device file, enables the packet
             *      interface, and registers signal IPI_PKT_SIG with the driver.
             */
            if  (  ( fd = ipi_open( "/dev/hpi3/d1dev" )  )  <  0  )  {
                  perror( "ipi_open" );
                  exit(-1);
             }
             ...   send a packet
             /*
              *    Poll the driver for a response packet.
              */
             bzero( pkt, HPI3_PKT_SIZE );

             ctl.pki_packet  = (word *)pkt
             ctl.pki_nbytes  = HPI3_PKT_SIZE;
             while (1)  {
                     sigoff();
                     reply = ioctl( fd, PKI_RECEIVE, &ctl );
                     if  ( !reply ||  (errno != EPKI_NO_PACKETS)  )
                              break;
```

```
                pause();
        }

        sigon();
           .
           .
        close(fd);
}
```

If no error has occurred, the `ioctl` return code will be 0. If an error has occurred, the error code will be one of the following codes (in decimal):

`EFAULT`	14	An address specified points outside the user's address space.
`EPKI_TOO_LARGE`	403	The response buffer is not large enough to accommodate the response packet.
`EPKI_NOT_ENABLED`	406	The packet interface has not been enabled.
`EPKI_IOCTL_REQT`	415	`ioctl` is not valid for the device type.
`EPKI_RESPBUF_LOST`	429	That part of the packet returned in a response buffer was lost.

The symbols for these error codes are located in file `errno.h`.

PKI_SEND

The `PKI_SEND` request is used to send request packets to the IPI-3/HIPPI packet driver.

The argument to the `ioctl` system call is a pointer to structure `pki_ctl`, described in the `sys/pki_ctl.h` file. The `pki_packet` field of structure `pki_ctl` must be set to a pointer to a valid IPI-3/HIPPI IOP request. The `pki_nbytes` field of structure `pki_ctl` must be set to the size of the IPI-3/HIPPI request packet. The size of the packet cannot exceed `EPAK_MAXLEN`.

The packet will be accepted only if the following conditions are met:

- The packet interface must be enabled
- The request cannot exceed the request limit
- The request code must be a valid EIOP request code
- The request must be valid for the device
- The IOP driver must already be started

The request packet must be of a format defined in the `sys/epackk.h` file.

You can issue this `ioctl` request only to IOP devices and IPI-3 devices.

You may use the C library routine `ipi_put_pkt` to send packets.

The following example shows how to send an IOP packet:

```
/*
 *   Send an iop packet
 */

#include <sys/types.h>
#include <sys/fcntl.h>
#include <errno.h>
#include <sys/epack.h>
#include <sys/epackk.h>
#include <sys/cnthpi3.h>
#include <sys/pki_ctl.h>

main()
{
            epackk  *pk;
            struct  pki_ctl    ctl;

            char  pkt[HPI3_PKT_SIZE];

            int         fd;

            /*
             *   Open the  IPI-3 device
             *     ipi_open is a library routine that registers for signal
             *     IPI_PKT_SIG, opens the device file, enables the packet
             *     interface, and registers signal IPI_PKT_SIG with the driver.
             */
            if  (  ( fd = ipi_open("/dev/hpi3/d1dev" )  )  <  0  )  {
                  perror( "ipi_open" );
                  exit(-1);
            }

            /*
             *   Create the IOP request
             */
            bzero( pkt, HPI3_PKT_SIZE );

            pk = (epackk *)pkt;
            pk->ek_open_stream.ek_request = EK_OPEN_STREAM;

            /*
             *    Send the request.
```

```
             */
            ctl.pki_packet  = (word *)pk
            ctl.pki_nbytes  = sizeof(ek_open_stream);

            if  (  ioctl( fd, PKI_SEND, &ctl )  <  0 ) {
                     perror( "Unable to send the packet");
            }

            close(fd);
}
```

If no error has occurred, the ioctl return code will be 0. If an error has occurred, the error code will be one of the following codes (in decimal):

EFAULT	14	An address specified points outside the user's address space.
EINVAL	22	The packet length is not in the range 1 through EPAK_MAXLEN. Or, on a request that requires a data transfer, the memory type is not valid.
EPKI_ASYNCH_LIM	404	Cannot enable more asynchronous responses than the maximum allowed.
EPKI_INVAL_CODE	405	Request code specified is not valid.
EPKI_NOT_ENABLED	406	Packet interface is not enabled.
EPKI_REQ_LIM	407	Exceeded the maximum number of requests allowed.
EPKI_BAD_RESYNC	408	A command list request was issued with a resynchronize code that does not match the resynchronize code of the last command list response.
EPKI_NO_START	409	The IOP driver has not been started.
EPKI_REQT_TYPE	414	Request code specified is not valid for the device type.
EPKI_IOCTL_REQT	415	ioctl is not valid for the device type.
EPKI_IOP_SEND	416	The packet driver could not send the packet to the IOP.
EPKI_DEVS_ACTIVE	418	Cannot stop an IOP driver that has an active device.
EPKI_CMDLIST_LIM	430	Exceeded the maximum number of command list requests allowed.

The symbols for these error codes are located in file errno.h.

PKI_SIGNO

The PKI_SIGNO request is used to register for a signal that will be sent to the user when the packet driver receives a response from the IOP.

The argument to the ioctl(2) system call is a pointer to structure pki_ctl, described in the sys/pki_ctll.h file. The pki_signo field of struct pki_ctl must be set to the signal number.

Before registering for a signal (PKI_ENABLE), you must enable packets.

You can issue this request only to IOP devices or IPI-3 devices.

The following example shows how to issue a request to register a signal by using the packet driver.

```
/*
 *    Register a signal by using the packet driver.
 */

#include <sys/types.h>
#include <sys/fcntl.h>
#include <sys/signal.h>
#include <errno.h>
#include <sys/pki_ctl.h>

main()
{
            struct  pki_ctl      ctl;

            int         fd;

            /*   Register for the packet-available signal   */

            if ( sigctl( SCTL_REG, IPI_PKT_SIG, 0 ) < 0 ) {
                   exit(errno);
            }

            /*     Open the  device  */

            if ( (fd = open("/dev/ipi3/d1dev", O_RDWR ) < 0 ) {
                    perror( "Unable to open the device" );
                    exit(errno);
            }

            . . . enable packets (see PKI_ENABLE)

            ctl.pki_psigno  =  IPI_PKT_SIG;
            if  (  ioctl( fd, PKI_SIGNO, &ctl )   <  0  )  {
                     perror( "Unable to register for a signal");
            }

            close(fd);
}
```

If no error has occurred, the `ioctl` return code will be 0. If an error has occurred, the error code will be one of the following codes (in decimal):

`EFAULT`	14	The `ioctl` argument specified points outside the user's address space.
`EINVAL`	22	The signal number specified is not within the range 0 to NSIG.
`EPKI_NOT_ENABLED`	406	The packet interface has not been enabled.
`EPKI_IOCTL_REQT`	415	`ioctl` is not valid for the device type.

The symbols for these error codes are located in file `errno.h`.

SEE ALSO

`hpi3_clear`(8), `hpi3_config`(8), `hpi3_option`(8), `hpi3_start`(8), `hpi3_stat`(8), `hpi3_stop`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

hpm – Hardware Performance Monitor interface

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

Cray PVP systems contains a Hardware Performance Monitor (HPM) that counts certain activities within a CPU when it is executing in user mode. The HPM driver interfaces let a user read the performance counters for the groups of processes or change to a different monitor group number. No group number change is supported on the CRAY C90 and CRAY T90 series.

The HPM driver supports the following three minor devices:

`/dev/hpm` Minor device 0; returns data about the current process.

`/dev/hpm_mult` Minor device 1; returns data about the current process and any other processes that are or have been in a multitasking group with the current process.

`/dev/hpm_all` Minor device 2; returns data about all processes (systemwide) that are running or have been disconnected.

The HPM driver supports the `ioctl`(2), `open`(2), and `read`(2) system calls. The `read`(2) system call returns only `sizeof(struct hpm)` bytes on minor devices 0 and 1, and `(sizeof(struct hpm)*NCPU)` bytes on minor device 2. If the requested number of bytes is fewer than this, an error is returned. If more than this is requested, the request is truncated.

On Cray PVP systems (except CRAY C90 and CRAY T90 series), the format of the `ioctl` request is as follows:

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

ioctl(fildes,HPMSET,group)
```

The only valid `ioctl` requests are `HPMSET` on minor device 0 and 2. Super-user permission is required to perform an `HPMSET` on minor device 2. `HPMSET` accepts as an argument the HPM group selected. Valid HPM groups are 0, 1, 2, or 3. `HPMSET` on minor device 2 changes the value of the global HPM group to the group specified. All processes that have not explicitly chosen an HPM group are put into the global group the next time they are connected. Group 1 is the default for the global group.

The possible errors for the system calls are as follows:

`EFAULT` Returned if the read buffer is not entirely within the user field.

`EINVAL` Returned if the selected group is outside the allowable range, if the read buffer is too small, or if an `ioctl` call other than `HPMSET` is issued.

ENXIO Returned if requested on other than the CRAY J90 series or if a minor device other than 0, 1, or 2 is requested on an open(2), close(2), or read(2), or if a minor device other than 0 or 2 is requested on an ioctl.

EPERM Returned if an HPMSET on minor device 2 is tried and the user does not have super-user permissions.

NOTES

The system always maintains the counters on any mainframe that supports the HPM. Except for the CRAY C90 and CRAY T90 series, a user program usually starts in group 1, although the group number is inherited from the parent and can be overridden by setting the global group.

On the Cray PVP systems (except for the CRAY C90 and CRAY T90 series), accounting of execution time for autotasked and microtasked programs does not charge for time spent waiting on a semaphore. The group number defaults to group 1 to gain this information. If an autotasked or microtasked program runs in a different HPM group, the system does not have the information necessary to avoid charging for wait-semaphore time. Therefore, running an autotasked or microtasked program with a group other than 1 shows nonrepeatable execution times for the program. On the CRAY C90 and CRAY T90 series, wait-semaphore time is always included.

The definition of what is counted differs between the CRAY C90 and CRAY T90 series and the CRAY J90 series. For more information, see the functional description manual for your mainframe.

FILES

/dev/hpm

/dev/hpm_all

/dev/hpm_mult

/dev/MAKE.DEV

/usr/include/sys/hpm.h

SEE ALSO

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

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

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

Guide to Parallel Vector Applications, Cray Research publication SG–2182

NAME

hsx – High-speed External Communications Channel interface

IMPLEMENTATION

All Cray Research systems except CRAY J90 series and CRAY EL series

DESCRIPTION

The hsx interface (or hsx driver) drives a High-speed External (HSX) Communications Channel. Application processes use the hsx driver by means of the standard UNICOS close(2), ioctl(2), listio(2), open(2), read(2), reada(2), write(2), and writea(2) system calls. Each of the special files in the /dev/hsx directory represents one input or output HSX channel.

By convention, HSX file names have the following format:

/dev/hsx*n*/i*xx*
/dev/hsx*n*/o*xx*

n Physical channel number

i Input channel

o Output channel

xx Logical channel number

For operation in one direction, only one channel must be opened. If the application uses both read and write operations, it must open both an input and an output channel. A read operation is valid only on an input channel; a write operation is valid only on an output channel. All HSX I/O is raw; the user process is locked in memory as data moves directly between the user buffer and the channel. User buffers must therefore be word-aligned, and their length must be a multiple of 8 bytes.

Dedicated and Shared HSX Channels

The hsx driver supports two modes of operation: dedicated and shared. The mode is assigned to each channel when it is configured. If a channel is dedicated, only one process can have that channel open at a time. If a channel is shared, several processes can use one HSX channel. The driver enforces a protocol on the shared channel; this allows it to determine the destination of each incoming message.

CRAY Y-MP systems that are configured to read and write the HSX channels with SSD solid-state storage buffers do not support shared channels. For more information on configuring HSX channels on Cray PVP systems, see *UNICOS Networking Facilities Administrator's Guide*, Cray Research publication SG–2304.

HSX ioctl Requests

Several ioctl requests are available. They have the following format:

```
#include <sys/hx.h>
ioctl (fildes, command, arg)
struct hxio *arg;
```

The valid `ioctl` requests are as follows:

`HXC_CLR` Sends a clear signal on the output channel. This request is restricted to dedicated channels.

`HXC_EXC` Sends an exception signal on the input channel. This request is restricted to dedicated channels.

`HXC_GET` Gets the current driver parameter settings; stores them in the `hxio` structure referenced by *arg*.

`HXC_SET` Sets driver parameters from the structure referenced by *arg*.

`HXC_WFI` Waits for interrupt: blocks until a clear signal (from an input channel) or exception signal (from an output channel) is received. If the channel signal has already been received, the request returns immediately. This request is restricted to dedicated channels.

The `hxio` structure contains the following fields:

`unsigned int flags` Flags that control channel operation. The *flags* field controls the driver's options. The bits in the flags field are defined as follows:

- `HXCF_ACM` Sets the alternative checkbyte mode (for an output channel) or disables SECDED (for an input channel).
- `HXCF_DBG` Sets debug mode. On write operations, the data is discarded. On read operations, the input data is undefined. In debug mode, the driver does everything except actual I/O on the HSX channel. HSX channel hardware does not have to be present for this mode to be used.
- `HXCF_DED` The channel is open in dedicated mode; no other processes may share the channel. If this flag is set, the value in the *path* field is meaningless. The `ioctl` request `HXC_GET` is used to set this flag. The `ioctl` request `HXC_SET` ignores this flag.
- `HXCF_DOWN` If this bit is set, the channel is unavailable to other users. If this bit is clear, the channel is available. When the channel is down, the driver returns the error `ENXIO`. Programs that reconfigure the channel should issue a `close` request immediately after the `ioctl` request, then reopen the channel before trying to do anything else. This bit is set in the `ioctl` request `HXC_SET` on a dedicated channel. `HXC_SET` is a read-only flag.

`int err` Detailed error status from the last request; for a list of HSX errors, see the MESSAGES section.

`int path` Logical path number for device (ignored on `HSC_SET`).

`unsigned int tmo` Time-out value (in seconds).

`listio` Special Features

The `hsx` driver defines two flags to be used in the `li_drvr` field of the `listreq` structure (see `listio`(2)):

`HXLI_CHD` Indicates, in effect, that user data is chained. When this flag is set, the driver suppresses the end-of-block signal at the end of the current list entry (for an output channel) or allows the current input data block to overflow into the next list entry (for an input channel).

`HXLI_STRB` Specifies that each stride (in memory) is a block. (A *stride* is the distance from the start of one section of data to the start of the next section.) This flag is valid only on dedicated channels and is meaningful only when the number of strides in a list entry is greater than 1. When the `HXLI_STRB` flag is set, `HXLI_CHD` is ignored.

If the driver encounters an end-of-block signal during a read operation in which the `HXLI_CHD` flag is set, no error indication is returned to the user. Subsequent read operations will complete with a data length of 0 until the `HXLI_CHD` flag is cleared. If the user issues the `ioctl` request `HXC_GET` to check the status after a chained read operation, the `err` field of the `hxio` structure is set to the error code `HXST_EOB`, indicating that an end-of-block signal arrived before the last read request was processed.

If the driver has not detected an end-of-block signal on a unchained read operation (that is, `read`(2), `reada`(2), or `listio`(2) with the `HXLI_CHD` flag clear) by the time the read operation completes, the driver discards the unread portion of the block. The byte count returned to the user is equal to the size of the buffer, and no error code is returned in `errno`. If the user issues the `ioctl` request `HXC_GET` to check the status after a unchained read operation, the `err` field of the `hxio` structure is set to the error code `HXST_LONG`, indicating that the entire block was not read and the remainder was discarded.

When using the `listio` chaining feature on shared channels to combine more than one buffer to make one data block, the user must include all buffers in the same `listio`(2) system call. That is, ensure that the `HXLI_CHD` flag is clear in the `li_drvr` field of the last item for an HSX channel in each `listio`(2) call. Failure to do this can cause lost or corrupt data on the channel if the user process is swapped between requests.

HSX Protocol

The `hsx` driver enforces no protocol on dedicated channels. On shared channels, the driver assigns a logical path value to each device configured on the shared channel. The `ioctl` request `HXC_GET` returns the logical path value for a shared channel in the *path* field of the `hxio` structure.

Each block of data must begin with a word that contains the logical path values of the sending and receiving channels. The first word must have the following format (as defined in the `sys/hx.h` include file):

```
struct hxhdr {
        unsigned int    unused  :32;    /* not used by driver   */
        unsigned int    to      :16;    /* destination address  */
        unsigned int    from    :16;    /* source address       */
};
```

The driver uses only two fields in the word. These fields have the following meanings:

`to` Identifies the channel to receive the message. The driver looks at this field in each incoming block and delivers the block to the process reading the channel assigned to the number in this field.

`from` Identifies the channel sending the message. In a message on an output channel, this field contains the value assigned at configuration time to the special file in `/dev` that represents the device.

You can use any protocol that fits this template on shared HSX channels. The sending process must set the `to` field in the `hxhdr` structure to the correct destination address. If no process is reading the destination device for incoming data, the driver discards the block after a period of time. During this time interval, no data can flow on the HSX channel.

Software Loopback Feature

The `hsx` driver contains a software loopback feature to debug application codes. The software loopback feature works on the `/dev` HSX files that are configured as HSX software loopback channels.

Loopback channels come in pairs; an even-numbered channel (*n*) is paired with an odd-numbered channel (*n*+1). Data written on the odd channel can be read on the even channel. A pair of channels simulates one set of HSX channels cabled in loopback configuration.

MESSAGES

The `hsx` driver returns one of the following error codes in `errno` on a fatal error. The error codes and their meanings are as follows:

`EFAULT` An `ioctl` request specified a bad argument address.

`EINVAL` The driver software has detected a fatal parameter error. Errors in system configuration and user errors in system call invocation can cause this error.

`EIO` One of the following conditions can cause this error:

- A fatal I/O error occurred or the HSX channel closed while asynchronous I/O was active (on a `read`(2) or `write`(2) system call).
- A data block was too long for the input buffer (on a `read`(2) system call) or overflowed the channel (on a `write`(2) system call).
- An I/O request timed out (on a `read`(2) or `write`(2) system call).

The detailed error status is available in the `err` field of the `hxio` structure; to read this field, use the `ioctl` request `HXC_GET`.

`ELATE` An input request timed out.

`ENXIO` The HSX channel is unavailable (on an `open`(2) system call), or the channel is not open (on a `close`(2) system call). This code can mean that the IOP did not allocate some resource.

After any fatal error, the detailed error status is retrieved by using the `ioctl` request `HXC_GET`. The driver returns error codes in the `err` field of the `hxio` structure.

`HXST_ABRT`	Channel abort (output only).
`HXST_BUF`	IOS I/O buffer unavailable on open operation.
`HXST_CHAN`	CPU gave bad channel number to IOS; caused by configuration error.
`HXST_CLR`	Clear pulse received (input channel only).
`HXST_DATA`	SECDED error or lost data (input only).
`HXST_DBG`	Debug mode error; indicates a driver fault (should never occur).
`HXST_EOB`	Unexpected end-of-block (input only).
`HXST_FLGS`	Buffer flags do not match; indicates a driver fault (should never occur).
`HXST_FMEM`	IOS free memory unavailable on open operation.
`HXST_FNC`	Illegal function code; indicates a driver fault (should never occur).
`HXST_HISP`	No high-speed channel to this IOP; caused by configuration error or wrong target memory.
`HXST_LLEN`	Transfer length too long; indicates a driver fault (should never occur).
`HXST_LONG`	Long block received (input only).
`HXST_MOS`	MOS buffer unavailable on open operation (debug mode only).
`HXST_NDEV`	No device present on the channel (hardware signal).
`HXST_OK`	No error (binary 0).
`HXST_OPEN`	Channel is not open; indicates a driver fault (should never occur).
`HXST_OVER`	Data-overrun error (input only).
`HXST_TM`	Bad target memory type; indicates a driver fault (should never occur).
`HXST_TMO`	Request timed out.
`HXST_XDT`	Exception pulse received during transfer (output only).
`HXST_ZLEN`	Buffer length is 0; indicates a driver fault (should never occur).

FILES

`/dev/hsx`*n*`/*`	HSX channel special files
`/usr/include/sys/hpacket.h`	
`/usr/include/sys/hx.h`	
`/usr/include/sys/hxsys.h`	

SEE ALSO

`close`(2), `ioctl`(2), `listio`(2), `read`(2), `reada`(2), `write`(2), `writea`(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

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

General UNICOS System Administration, Cray Research publication SG–2301

NAME

hy – HYPERchannel adapter interface

IMPLEMENTATION

Cray PVP systems except CRAY J90 series

DESCRIPTION

The HYPERchannel driver provides an interface for NSC HYPERchannel adapters connected to the IOS. The HYPERchannel driver also is used for Cray Research front-end interfaces (FEIs) and VME interfaces attached to the IOS. For more information, see `fei`(4) and `vme`(4). The driver accepts standard UNICOS `close`(2), `listio`(2), `open`(2), `read`(2), `reada`(2), `write`(2), and `writea`(2) system calls; it also accepts `ioctl`(2) requests.

By convention, the HYPERchannel special file names are in the following format:

`/dev/comm/`*address*`/lp`*nn*

address The address is composed of three elements: the interface type, the IOS number, and the channel number (in octal). The interface type is indicated by one character: `f` (FEI), `n` (NSC), or `v` (VME). The IOS number is either 0 or 1. The channel number is the octal channel to which the interface is connected on the specified IOS.

nn Logical path number for the device (0 through 15).

For each device, up to 16 logical paths are available. The minor number is the logical path across a device. For example, if the site has configured 3 network interfaces that use the `hy` driver with 16 logical paths for each, the first device in the `comm_info` of type `COMM_HYDRIVER` uses logical paths 0 through 15. The second device uses logical paths 16 through 31. The third device uses 32 through 47.

You cannot open a minor device when it is already open. Any attempt to do so fails with an `EBUSY` error.

You can use the `ioctl` requests `HYGET` and `HYSET` to change parameters for the IOS. Kernel-level routines can use the `HYKGET` and `HYKSET ioctl` requests to change parameters for the IOS. To return various kinds of driver and HYPERchannel status, use the `HYSTAT` function. The subfunctions are defined in the `npstat` structure in `sys/hy.h`. The last status is the IOS driver's status and is specific to that driver. The `HYHLTIO ioctl` request allows users to halt outstanding driver packet I/O requests. Because this request must be issued by the same process that issued the I/O, you can use it only with asynchronous I/O.

The `ioctl` structure is defined in the `sys/hy.h` include file, as follows:

```
struct npstat {
        int     mad;    /* maximum associated data size        */
        int     wtmo;   /* write timeout                       */
        int     rtmo;   /* read timeout                        */
        int     inq;    /* input messages to queue             */
        int     path;   /* adapter path requested              */
        int     lrt;    /* last response time                  */
        int     nperr;  /* last N packet error return from IOS */
                        /* (valid only when the last operation */
                        /* returned an error)                  */
        int     sfunc;  /* status subfunction see npstats.h    */
        char    *sbuf;  /* status buffer pointer               */
} ;
```

`mad` Maximum associated data size (in words); this parameter places a limit on the message size that the IOS expects.

`wtmo` Write time-out.

`rtmo` Read time-out (in tenths of a second) for both read and write operations. The IOS queues up to four messages before discarding input from the adapter that the mainframe has not read.

`inq` Input messages to queue; allows a process to specify up to four messages to queue. A process can request a specific minor device number. This is currently unused.

`path` Adapter path.

`lrt` Last response time; the time since the last packet was received from the IOS for this minor device.

`nperr` Last error status from the IOS; valid only when the last I/O operation returns an error. For a list of error response codes from the IOS and their meaning, see the `nperr` Values subsection.

`sfunc` The statistics subfunction to be used; used only in conjunction with the `HYSTAT ioctl` request.

`sbuf` The address of the driver statistics buffer; used only in conjunction with the `HYSTAT ioctl` request.

You can use an `ioctl` request after the `open` operation and before the first `read` or `write` operation to change the IOS parameters.

The `read`(2) and `write`(2) system calls issued by a process to the HYPERchannel driver differ from typical `read`(2) and `write`(2) system calls only in that the first 64 bytes of the data must be a HYPERchannel message proper. The message proper is defined as follows:

```
struct mp {
        char    control[2];     /* NSC control word           */
        char    acode[2];       /* NSC access code            */
        char    to[2];          /* NSC destination adapter    */
        char    from[2];        /* NSC source adapter         */
        char    param[56];      /* NSC parameters             */
} ;
```

The details of the contents of message proper fields are available in NSC documentation.

If a `read`(2) system call is not satisfied within the time-out period, an `ELATE` error is returned. If a `write`(2) system call cannot be completed, it is retried periodically in the time allowed by the time-out period before an `ELATE` error is returned. A `close`(2) system call closes the minor device. A `CEIO` error terminates any outstanding system calls.

`nperr` Values

The following tables list the possible values of `nperr`, according to IOS driver (a response of 0 always means "no error").

The following octal status values have the associated meanings for all N-packet drivers:

Value	**Definition**
`03`	Protocol error concerning N-packet request order.
`04`	Bad channel, owner, or path.
`05`	Bad function code.
`11`	Cannot create activity; channel, path, or memory not available.
`12`	Error on configuration request processing.
`54`	Path terminated.

The following octal status values have the associated meanings only for FEI drivers:

Value	**Definition**
`40`	Message length is 0 or too big.
`41`	Read time-out.
`43`	Write time-out.
`44`	Write sequence error.
`50`	Illegal function or subfunction.
`52`	MOS or local memory not available; cannot create.

53	Port select error.
54	Driver terminating.
56	Message read is too short; data read is too big.
57	Insufficient space allocated in CPU for input.
60	Parity error received.
61	Read sequence error.

The following octal status values have the associated meanings only for VME and NSC drivers:

Value	**Definition**
36	Path 0 not available for loopback destination.
41	Read time-out.
43	Write time-out.
45	No read for loopback write.
51	No remote adapter ID in write message.
52	MOS or local memory not available.
57	Insufficient space allocated in CPU for input.
50	Illegal function or subfunction.
56	Transfer length specified on write is not valid.
60	Loopback write error.
62	Bad CPU address given in N-packet.

The following octal status values have the associated meanings only for VME drivers:

Value	**Definition**
42	Input channel time-out.
44	Output sequence error.
52	Cannot create activity.
53	Output channel time-out.
57	Parity error, message bad size, or data not present but expected.

The following octal status values have the associated meanings only for NSC drivers:

Value	**Definition**
42	Read error.
44	Associated data flag is set, but no associated data is available.
53	Remote adapter not available; write aborted.

61	Residual data on channel after data read.
63	Input sequence error.
64	Input parity error.

The following octal status values have the associated meanings for COMM drivers (12-Mbyte interconnect specification drivers):

Value	**Definition**
20	CPU I/O request time-out.
21	Driver detected function code error.
22	Driver termination in progress.
23	Insufficient space allocated by CPU.
30	Parameter block detected is not valid.
31	Data detected is not valid.
32	Write time-out.
33	Driver detected parity or sequence error.
34	I/O channel time-out.
35	Overrun (input channel).
36	Transfer length error.

BUGS

The NSC status is not available from the IOS. Currently, no special functions of the adapter are supported.

FILES

`/dev/comm/*/*`

`/usr/include/sys/hy.h`

`/usr/include/sys/hysys.h`

SEE ALSO

`fei`(4), `vme`(4)

`ioctl`(2), `listio`(2), `read`(2), `reada`(2), `write`(2), `writea`(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

NAME

`inode` – inode file system

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `/inode` file system allows privileged processes access to a file or directory when the process knows the device and inode number of a file system. The `/inode` file system has no files or storage but if it is asked to translate a path of the form `/inode/`*ddd.iii.ggg.ff* (where *ddd* is the device number, *iii* is the inode number, *ggg* is the optional generation number, and *ff* is an optional list of flags `r` or `w`), it returns the vnode for the specified file or directory. If the generation number is provided it must match the current generation number.

Access to the */inode* path translation is limited to privileged processes: Root user for systems with traditional UNIX security or users with an active `secadm` category for systems using privilege assignment lists (PALs).

Executing an `ls`(1) or `readdir`() on the `/inode` file system root directory shows only the "." and ".." directories.

Since `/inode` has no storage or files but serves only as a path to other file systems, once the path element following the `/inode` is translated, the normal rules for file system access for that target item apply.

CONFIGURATION

You create an `/inode` file system with the following steps:

1. Create an empty directory called `/inode` with the following command:

   ```
   mkdir /inode
   ```

2. Modify the system mount scripts and `/etc/fstab` file so that the `/inode` file system is always mounted at system startup.

   `fstab` entry:

   ```
   /inode /inode INODE
   ```

NOTES

The use and implementation of the `ioctl`(2) operations documented in this entry are subject to change in future releases of UNICOS.

FILES

`/inode` `/inode` file system root

SEE ALSO

`fstab`(5)

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

NAME

iosE – IOS model E interface

IMPLEMENTATION

Cray PVP systems with an IOS model E

DESCRIPTION

The /dev/iosE/* device is used as the interface to the IOS model E (IOS-E). The ioctl(2) system call issues requests to the ioscntl device driver.

The minor number of a device specifies a particular IOP (for example, the /dev/iosE/iop.1.3 device has a minor number of 11, which indicates cluster 1, eiop 3).

The different structures used for the ioctl(2) system call are defined in the /usr/include/sys/ioscntl.h include file, and they are described as follows:

```
#define get_cluster(x)  (int)(((uint)x>>3)&0377)  /* get cluster from minor */
#define get_eiop(x)     (int)(x&07)               /* get eiop from minor    */

/*
 *     Ioctl argument structure
 */
struct ioscntl {
        union {
        int i;
                char *cp;
                word *wp;
                struct ioscntl_cf *cf;
        } ios_arg1;
        union {
                int i;
                word *wp;
        } ios_arg2;
        int ios_arg3;
        int ios_arg4;
};
/* Field equivalence defines */
#define      ios_hisp_no     ios_arg1.i
#define      ios_channel     ios_arg2.i
#define      ios_mode        ios_arg3
#define      ios_target      ios_arg4
#define      ios_path        ios_arg1.cp
#define      ios_send        ios_arg1.wp
#define      ios_receive     ios_arg2.wp
#define      ios_length      ios_arg3
#define      ios_iop_config  ios_arg1.cf
/*
```

```
 *     IO_GET_CONFIG buffer structure.
 */
struct ioscntl_cf {
        int ios_status;                 /* configuration status bits         */
        char ios_bootpath[IO_PATHMAX];  /* partial path name for boot binary */
        int ios_usage;                  /* drivers currently using IOP       */
        /* the following fields are valid only for the MUXIOP                */
        int ios_lowsp;                  /* low-speed channel number          */
        struct {
                int open;               /* open flag                         */
                int channel;            /* channel number                    */
                int target;             /* the target memory for the channel */
                int mode;               /* operating mode for the channel    */
        } ios_hisp[2];
};
```

Configuration requests affect future I/O in the following ways:

`IO_STOP` New I/O to the target IOP is queued but not processed. Current I/O is not guaranteed to complete.

`IO_ABORT` All current and new I/O to the target IOP is rejected.

`IO_RESTART` Any current I/O that has not completed is requeued. All queued I/O to the target IOP is then processed.

The following is a description of available `ioctl` requests:

`IO_SET_PATH` Specifies the path name for a binary file that will be booted later into the target IOP by using the `IO_BOOT_IOP` request. This request can be done only to a IOP that has not been booted.

Required argument:

ios_path Pointer to the path name of the boot binary file

`IO_SET_LOWSP` Sets the low-speed channel number. This request can be done only when the low-speed channel is down.

Required argument:

ios_channel Low-speed channel number

`IO_SET_HISP` Sets the HISP channel parameters. This request can be done only when the MUXIOP for the specified cluster is up and accessible through the low-speed channel, and the HISP is down.

Required arguments:

ios_hisp HISP ordinal; 0 for HISP0, 1 for HISP, and so on.

ios_channel HISP channel number.

ios_mode HISP channel mode.

ios_target HISP channel target memory.

`IO_BOOT_IOP` Boots the specified IOP. The binary file used to boot the IOP must have been specified by a previous `IO_SET_PATH` request.

`IO_DOWN_IOP` Sets the specified IOP to a down state. This causes the `IO_ABORT` condition to be set for the target IOP.

`IO_RDOWN_IOP` Sets the specified IOP to a down state in a restartable way. This causes the `IO_STOP` condition to be set for the target IOP.

`IO_UP_LOWSP` Sets the low-speed channel to an up state. This request can be done only when the low-speed channel is down. This causes the `IO_RESTART` condition to be set for those IOPs in the cluster that were stopped by a previous `IO_DOWN_LOWSP` request.

`IO_DOWN_LOWSP` Sets the low-speed channel to a down state. This request can be done only when the low-speed channel is up. This causes the `IO_STOP` condition to be set for those IOPs in the cluster that are processing requests.

`IO_UP_HISP` Sets a HISP channel to an up state. This request can be done only when the HISP channel is down.

Required argument:

ios_hisp HISP ordinal; 0 for HISP0, 1 for HISP, and so on.

`IO_DOWN_HISP` Sets a HISP channel to a down state. This request can be done only when the HISP channel is up.

Required argument:

ios_hisp HISP ordinal; 0 for HISP0, 1 for HISP, and so on.

`IO_UP_INTER` Sets an inter-IOP channel to an up state. This causes the `IO_RESTART` condition to be set for IOP if it was processing requests at the time of a previous `IO_DOWN_INTER` request.

`IO_DOWN_INTER` Sets an inter-IOP channel to a down state. This causes the `IO_STOP` condition to be set for IOP if it is processing requests.

`IO_ECHO` Echoes message through an IOP.

Required arguments:

ios_send Word pointer to the send buffer

ios_receive Word pointer to the receive buffer

ios_length Number of words in the echo message

`IO_GET_CONFIG` Gets an IOP configuration.

Required argument:

ios_iop_config Pointer to the buffer structure to receive the data

In addition to the standard `ioctl` error codes (see `ioctl`(2)), the following errors cause an `ioctl` request to fail:

`[EFAULT]`	Bad address passed to system call.
`[EAGAIN]`	Attempt to down an inter-IOP channel that is already down.
`[ENXIO]`	MUXIOP or IOP is not in a state that allows configuration of an inter-IOP channel.
`[EBUSY]`	MUXIOP or IOP is currently processing another `ioctl`(2) request.

NOTES

Only the super user can use the `/dev/ios` interface.

FILES

`/dev/ios`

`/usr/include/sys/epack.h`

`/usr/include/sys/ioscntl.h`

SEE ALSO

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

NAME

ipi3 – IPI-3/IPI interface

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

The IPI-3 interface is used as a connection to the IPI-3/IPI IOPs and the IPI-3/IPI devices. These devices are represented by special files in the /dev/ipi3 directory.

The ASCII names for the IOP devices and the IPI-3/IPI devices in the /etc/config/ipi3_config configuration file are used to create the special files in /dev/ipi3.

The pki_ctl structure, as defined in the sys/pki_ctl.h include file, is used to communicate between the packet driver and the controlling process. The packet driver control structure is defined as follows:

```
struct pki_ctl{
       int    pki_psigno;                /* Signal to receive        */
       word   *pki_packet;               /* Packet from user program */
       int    pki_nbytes;                /* Length of packet         */
       int    pki_device;                /* Device name              */
       }
```

The following is a list of the ioctl(2) requests used with the IPI-3/IPI interface:

PKI_CLEAR	Clears the IPI-3/IPI device.
PKI_DRIVER_STS	Returns the status of the IPI-3/IPI packet driver.
PKI_ENABLE	Enables a packet interface.
PKI_GET_CONFIG	Returns the IPI-3/IPI configuration.
PKI_GET_DEVCONF	Returns the configuration of a device.
PKI_GET_DEVTBL	Returns a driver device table.
PKI_GET_OPTIONS	Returns the options of the IPI-3/IPI packet driver.
PKI_RECEIVE	Returns an IPI-3/IPI packet.
PKI_SEND	Sends an IPI-3/IPI packet.
PKI_SET_OPTIONS	Modifies the options of the IPI-3/IPI packet driver.
PKI_SIGNO	Registers the signal to be sent to the user when an interrupt is received from the IOP.

FILES

/dev/ipi3/reqt	IPI-3/IPI interface device
/usr/include/sys/pki_ctl.h	Structure definition of pki_ctl

SEE ALSO

reqt(4)

ipi3_clear(8), ipi3_config(8), ipi3_option(8), ipi3_start(8), ipi3_stat(8), ipi3_stop(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

Tape Subsystem Administration, Cray Research publication SG–2307

NAME

ldd – Logical disk device

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

The files in /dev/ldd are logical disk descriptor files (see ldesc(5)). They are used to combine other logical or physical disk slices into a single logical disk device. Typically, the files in /dev/ldd are referenced by name by a disk block special file in /dev/dsk that has a major device number of the concatenated logical disk driver, ldd. See dsk(4). Concatenated logical disk devices are differentiated from other logical disk devices by the major device number defined in /usr/src/uts/c1/cf/devsw.c.

Usually, a logical descriptor file is used to combine physical disk slices in the following manner (see pdd(4)):

```
mknod /dev/ldd/usr L /dev/pdd/usr.0 /dev/pdd/usr.1
```

When the /dev/ldd/usr file is referenced by a character or block special device, its member slices, /dev/pdd/usr.0 and /dev/pdd/usr.1, are combined by a logical disk driver into a single logical disk device.

Typically, a block special file in /dev/dsk references a file in /dev/ldd. Following through with the preceding example, you can make a simple concatenated logical device by using the mknod(8) command (see mknod(8) and dsk(4)). In this example, the major device number is dev_ldd, and the minor device number is 12. The two 0's are placeholders and are unused.

```
mknod /dev/dsk/usr b dev_ldd 12 0 0 /dev/ldd/usr
```

The following logical disk devices are supported:

dev_ldd — The simple concatenated logical device. The physical slices are concatenated to form a single logical disk device. See dsk(4).

dev_mdd — A mirrored logical disk device. Two or more disk slices are identical copies of one another for data redundancy. See mdd(4).

dev_sdd — A striped logical disk device. Two or more disk slices are striped to increase bandwidth. See sdd(4).

The logical disk devices are differentiated by their major device numbers defined symbolically in /usr/src/uts/c1/cf/devsw.c.

FILES

`/dev/dsk/*`

`/dev/ldd/*`

`/usr/include/sys/ldesc.h`

SEE ALSO

`dsk`(4), `ldesc`(5), `mdd`(4), `pdd`(4), `sdd`(4), `ssdd`(4)

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

NAME

`lo` – Software loopback network interface

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The loop interface, `lo`, is a TCP/IP pseudo device. It is a software loopback mechanism, which you can use for performance analysis, software testing, and/or local communication. By default, the loopback interface is accessible at address 127.0.0.1. To change this address, use the `ioctl`(2) request `SIOCSIFADDR`.

SEE ALSO

`inet`(4P)

NAME

log, klog – System message log files

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The /dev/log and /dev/klog special files contain messages for the syslogd(8) system log daemon. The /dev/log file is the user-level system log; it contains the messages issued by syslog(3C). The /dev/log file is a FIFO special file (named pipe).

The /dev/klog file is the kernel-level system log; it contains the kernel messages from the system log daemon, syslogd(8). The /dev/klog file is a circular queue; the kernel writes kernel messages into it, and syslogd(8) reads kernel messages from it.

NOTES

Only the syslogd(8) utility should read /dev/log and /dev/klog. When two or more processes have /dev/log or /dev/klog open at the same time, results are undefined.

FILES

/dev/klog	Kernel-level system log
/dev/log	User-level system log

SEE ALSO

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

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

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

NAME

mdd – Mirrored disk driver

IMPLEMENTATION

Cray PVP systems with an IOS model E

DESCRIPTION

The files in /dev/mdd are character special files that allow read and write operations to mirrored disk slices. A *mirrored disk slice* is a logical disk device composed of two or more physical disk slices. These physical slices, also known as *members*, must be of the same length and physical sector size. A maximum of MDDSLMAX (defined in the parameter file) mirrored devices can exist.

A mirrored disk device takes on the physical sector size of its member physical disk devices. If the sector size of the member devices consists of more than 512 words, the I/O request lengths and starts must match those for the specified device.

Device driver-level mirroring is used when reliability is a critical issue. An individual write I/O request is sent to all members of the mirror that are write enabled. Read I/O requests are sent to the first read-enabled device of the mirrored group that is not busy.

On read errors, the driver selects the next available device from which to read.

On write errors, the member in error is disabled, and no reads or writes go to that device. The node also is marked showing that the device is now out of sync with the other mirrored members. This marking is recorded externally in the /dev/mdd/*name* node for the device, permitting the information to be preserved across reboots. To resynchronize, use the mddconf(8) and mddcp(8) commands.

The files in /dev/mdd are not usually mountable as file systems. You may specify a mirrored disk slice as a whole or part of a logical disk device. The files in /dev/mdd are all of the *logical indirect* type. See dsk(4), ldesc(5), ldd(4), and pdd(4).

The mknod command is used as follows to create a mirrored disk inode:

mknod *name type major minor* 0 *rwmode path*

name Name of the logical device.

type Type of the device data being transferred. Devices in /dev/mdd are character devices denoted by a c.

major Major device number of the mirrored logical disk device driver. The driver is denoted by the name dev_mdd in the /usr/src/uts/c1/cf/devsw.c file.

minor Minor device number for this slice. Each striped disk slice must have a unique minor device number.

0 Placeholder for future use.

rwmode Read/write/initialize modes, in the form `0`*xx*. Reading from right to left, each digit represents a member of mirrored group. The bits represent read enable, write enable, and initialize, and they also are read from left to right, as permissions on a file are read. For example, 037 is a two-member mirror: member 0 is read/write/initialize; member 1 is write only and initialize.

path Path name that designates the logical descriptor file listing the member slices. See `ldesc`(5).

NOTES

As noted above, an unsynchronized mirrored device is marked in the `/dev/mdd/`*name* node. If these nodes are recreated, causing the loss of synchronization information, the device must be resynchronized manually.

EXAMPLES

The following example creates a mirrored disk inode:

```
mknod /dev/mdd/usr c dev_mdd 2 0 077 /dev/ldd/usr.mirror
```

FILES

`/dev/mdd/*`

`/usr/include/sys/mdd.h`

`/usr/src/c1/io/mdd.c`

SEE ALSO

`dsk`(4), `ldd`(4), `ldesc`(5), `pdd`(4)

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

NAME

`mem`, `kmem` – Common or main memory files

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `mem` file is a special file that is an image of the common or main memory of the computer. It can be used to examine or patch the system. References to nonexistent locations fail with an `errno` of `ENXIO`.

The `kmem` file is the same as `mem`; `kmem` has been preserved, because physical memory and kernel memory are not exactly the same on some machines not manufactured by Cray Research.

You also can use the `mem` (or `kmem`) file to change the configuration of physical memory. Physical memory is made up of the following areas:

Kernel memory — Three separate areas that are not necessarily contiguous:

- Space allocated at build time
- Space allocated at boot time
- Space allocated at run time

RAM disk — Memory-resident disk.

User memory — Area in which user processes reside.

Guest memory — Area in which guest operating systems reside (mutually exclusive from downed memory).

Downed memory — Area that was formerly used by diagnostics. This area is an artifact of the archaic form of `chmem` (mutually exclusive from guest memory).

Maintenance memory — Area used by diagnostics.

The following `ioctl`(2) requests are defined in the `sys/mm.h` include file:

`MM_STATUS_ONLY` — Returns memory status.

`MM_RSV_MAINT` — Reserves maintenance memory by reducing the overall size of configured physical memory by the amount required for diagnostics; the required space comes from user memory.

On completion, if requested, the memory status is returned.

`MM_RLS_MAINT` — Releases maintenance memory by restoring the overall size of configured physical memory; the space used for maintenance memory is returned back to user memory.

On completion, if requested, the memory status is returned.

The value of the `ioctl` argument *arg* is the address of the following structure, which is defined in `sys/mm.h`:

```
struct mmioctl {
        struct mmstat  *mmi_stat;       /* memory status area (user
                                           relative) */
        int             mmi_statlen;    /* length of mem status area
                                           (bytes) */
};
```

If `mmi_statlen` is 0, no memory status will be returned.

Memory status (that is, the current configuration of physical memory) is returned by using the following structure. A `ba` suffix refers to base address, and an `sz` suffix refers to size. All units are in words.

```
struct mmstat {
        long    mms_flags;      /* memory state flags */
        long    mms_cnfphyssz;  /* configured physical memory size*/
        long    mms_actphyssz;  /* actual physical memory size */
        long    mms_kbuildba;   /* kernel allocated at build time */
        long    mms_kbuildsz;
        long    mms_kbootba;    /* kernel allocated at boot time */
        long    mms_kbootsz;
        long    mms_krunba;     /* kernel allocated at run time
                                   (i.e., kmem) */
        long    mms_krunsz;
        long    mms_ramba;      /* ramdisk */
        long    mms_ramsz;
        long    mms_usrba;      /* user memory */
        long    mms_usrsz;
        long    mms_plockdsz;   /* amount of user memory that's
                                   plocked */
        long    mms_downedba;   /* downed memory */
        long    mms_downedsz;
        long    mms_maintba;    /* maintenance memory */
        long    mms_maintsz;
};
```

In addition to the standard `ioctl` error codes (see `ioctl`(2)), the following are errors that cause an `ioctl`(2) request to fail:

`EACCES` Calling process was not plocked in memory when trying (possibly indirectly) to reduce the size of user memory

`EAGAIN` Maintenance memory is disabled while downed or guest memory exists

`EBUSY` Cannot idle system (that is, park all CPUs) so that memory can be reconfigured

`EDOM`	User's and kernel's idea of memory status structure size differ
`EFAULT`	Cannot copy information from or to user area
`EINTR`	Interrupted system call
`EINVAL`	Unrecognized request or device minor number is not 0, 1, or 2
`ENOSPC`	Space for maintenance memory is unavailable from user memory
`ENOSYS`	Maintenance memory is not supported

FILES

`/dev/MAKE.DEV`
`/dev/kmem`
`/dev/mem`

NAME

mnu – Interactive mnu-based display package

SYNOPSIS

/usr/src/uts/cmd/disk/mnu.c

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

The mnu program is a display package that provides a menu-based interactive command processing capability. An application program that uses the mnu program provides a set of linked menus in a structured form and is compiled with mnu. The application calls a function in mnu to update the display and to solicit input from users.

The mnu program provides command input and screen refresh capabilities by using the following primitives:

<TAB> or right arrow	Menu right
<BACK SPACE> or left arrow	Menu left
<RETURN>	Next menu level or execute menu item
<ESC>	Back to first menu level
<CONTROL-F> or <PAGE DOWN>	Next display page
<CONTROL-B> or <PAGE UP>	Previous display page
<CONTROL-D> or down arrow	Display down one line
<CONTROL-U> or up arrow	Display up one line
?	Help

The first letter of a given menu item selects and executes that menu item. A help facility displays help text for each menu item if provided by the application.

An application builds and links together menu items by using the mnu structure defined in /usr/include/sys/mnu.h.

```
/*
 *  menu structure
 */
struct mnu {
    char            *mu_name;       /* menu name */
    struct mnu      *mu_forw;       /* next menu level */
    int             (*mu_func)();   /* function to execute */
    int             mu_flags;       /* flags defined below */
    char            *mu_help;       /* pointer to help text */
};

/*
 *  menu flags
 */
#define MUF_INPUT   1               /* input mode */
#define MUF_STAY    2               /* stay on this menu */
```

The mu_name field is the name of the menu item. The mu_forw field points to the next level menu structure and is NULL at the bottom menu. The mu_func field is a pointer to a function, on the bottom menu item, that performs the desired action. The mu_flags control menu displays action, and the mu_help field is a pointer to optional help text.

The application program must provide an external mnu entry called mnu0 that points to the main (top) menu. The application calls mnu_refresh to update the display and menu items at the specified refresh rate.

The hddmon(8) utility provides an example of an application that makes use of mnu display and command capabilities. See /usr/src/uts/cmd/disk/hddmon.c.

FILES

/usr/src/uts/cmd/disk/hddmon.c

/usr/src/uts/cmd/disk/mnu.c

/usr/include/sys/mnu.h

SEE ALSO

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

NAME

etc/netconfig – Network configuration database

IMPLEMENTATION

Cray Research systems licensed for ONC+™ and UNICOS 8.3 or later

DESCRIPTION

The network configuration database, /etc/netconfig, is a system file used to store information about networks that are connected to the system. The network selection component also includes the NETPATH environment variable and a group of routines that access the network configuration database by using NETPATH components as links to the netconfig entries.

The netconfig database contains an entry for each network available on the system. Entries are separated by newlines. Fields are separated by white space in a prescribed order. You can embed white space as a blank single space or a tab symbol. You may embed backslashes (\) as symbols. Lines in /etc/netconfig that begin with a # symbol in column 1 are treated as comments.

Each of the valid lines in the netconfig database correspond to an available transport. Each entry is of the following form and order:

- *network ID*
- *semantics*
- *flag*
- *protocol family*
- *protocol name*
- *network device*
- *translation libraries*

network ID — A string that uniquely identifies a network. *network ID* consists of nonnull characters, and it has a length of at least 1. No maximum length is specified. This namespace is locally significant and is named by the local system administrator. All network IDs on a system must be unique.

semantics — The *semantics* field is a mandatory field that contains a string that identifies the semantics of a network. Semantics is defined as the services a network supports and the service interface the network provides. The following semantics are recognized.

tpi_clts	Transport Provider Interface (connectionless).
tpi_cots	Transport Provider Interface (connection oriented).
tpi_cots_ord	Transport Provider Interface (connection oriented) it supports an orderly release.

flag The *flag* field records two-valued (true and false) attributes of networks. *flag* is a string composed of a combination of symbols, each of which indicates the value of the corresponding attribute. If a specified symbol is present, the attribute is true. If a symbol is absent, the attribute is false. The `-` symbol indicates that none of the attributes are present. Only one symbol is currently recognized:

`v` Visible (default) network. Used when the `NETPATH` environment variable is not set.

protocol family The *protocol family* and *protocol name* fields are provided for protocol-specific applications. The *protocol family* field contains a string that identifies a protocol family. The *protocol family* identifier follows the same rules as those for network IDs; the string consists of nonnull characters, it has a length of at least 1, and no maximum length is specified. A `-` symbol in the *protocol family* field indicates that no protocol family identifier applies (the network is experimental). The following are examples:

`loopback` Loopback (local to host)

`inet` Internetwork: UDP, TCP, and so on

protocol name The *protocol name* field contains a string that identifies a protocol. The *protocol name* identifier follows the same rules as those for network IDs; that is, the string consists of nonnull characters, it has a length of at least 1, and no maximum length is specified. A `-` symbol indicates that none of the names listed apply. The following protocol names are recognized.

`tcp` Transmission Control Protocol

`udp` User Datagram Protocol

network device The *network device* field is the full path name of the device used to connect to the transport provider. The following network devices are recognized.

`/dev/tcp` Transmission Control Protocol

`/dev/udp` User Datagram Protocol

translation libraries
The name-to-address *translation libraries* field support a name-to-address mapping service and directory service for the network. A `-` in this field indicates the absence of any translation libraries, in which case, name-to-address mapping for the network is nonfunctional. This field consists of a comma-separated list of path names to libraries. Although this is not used in the UNICOS software, the path should be present.

Each field corresponds to an element in the `struct netconfig` structure. `struct netconfig` and the identifiers described previously are defined in <`netconfig.h`>. This structure includes the following members:

`char *nc_netid`	Network ID, including null terminator
`unsigned long nc_semantics`	Semantics
`unsigned long nc_flag`	Flags

`char *nc_protofmly`	Protocol family
`char *nc_proto`	Protocol name
`char *nc_device`	Full path name of the network device
`unsigned long nc_nlookups`	Number of directory lookup libraries
`char **nc_lookups`	Names of the name-to-address translation libraries
`unsigned long nc_unused[9]`	Reserved for future expansion

The `nc_semantics` field takes the following values, corresponding to the semantics identified previously:

- `NC_TPI_CLTS`
- `NC_TPI_COTS`
- `NC_TPI_COTS_ORD`

The `nc_flag` field is a bit field. The `NC_VISIBLE` bit, corresponding to the attribute identified previously, is currently recognized.

`NC_NOFLAG` indicates the absence of any attributes.

WARNINGS

You should not modify the `/etc/netconfig` file provided by Cray Research, because incoherent behavior of `rpcbind`(8) may result.

FILES

`netconfig.h`

SEE ALSO

`nsswitch`(4)

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

NAME

nis – A new version of the network information service

IMPLEMENTATION

Cray Research systems licensed for ONC+™ and UNICOS 8.3 or later

DESCRIPTION

NIS+ is a new version of the network information service. This version differs in several significant ways from version 2, which is referred to as NIS or YP in earlier releases. Specific areas of enhancement have been the ability to scale to larger networks, security, and the administration of the service.

The man pages for NIS+ are broken up into two basic categories. Section 8 man pages are user commands and daemons. Section 3N man pages describe the NIS+ programming API.

All commands and functions that use NIS version 2 are prefixed by the letters `yp` as in `ypcat`(1). Commands and functions that use the new version are prefixed by the letters `nis`, as in `nismatch`(8) and `nis_add_entry`(3N).

This man page introduces NIS+ terminology. It also describes the NIS+ namespace and authentication and authorization policies.

NIS+ Namespace

The naming model of NIS+ is based on a tree structure. Each node in the tree corresponds to an NIS+ object. There are six types of NIS+ objects:

- Directory
- Table
- Group
- Link
- Entry
- Private

NIS+ directories

Each NIS+ namespace will have at least one NIS+ directory object. An NIS+ directory is like a UNIX file system directory that contains other NIS+ objects, including NIS+ directories. The NIS+ directory that forms the root of the NIS+ namespace is called the *root directory*. Two special NIS+ directories exist: `org_dir` and `groups_dir`. The `org_dir` directory consists of all systemwide administration databases, such as `passwd`, `hosts`, and `groups`. The `groups_dir` directory consists of NIS+ group objects that are used for access control. The collection of the `org_dir`, `groups_dir` and their parent directory is referred to as an *NIS+ domain*. You can arrange NIS+ directories in a tree-like structure allowing the NIS+ namespace to be divided so that it matches an organizational hierarchy.

NIS+ tables

NIS+ tables, contained within NIS+ directories, store the actual information about some particular type. For example, the hosts system table stores information about the IP address of the hosts in that domain. NIS+ tables have multiple columns and any of the columns can be searched. Each table object defines the schema for its database.

NIS+ group objects

NIS+ group objects are used for access control at group granularity. NIS+ group objects, contained within the `group_dir` directory of a domain, contain a list of all NIS+ principals within a certain NIS+ group.

NIS+ link objects

NIS+ link objects are similar to UNIX symbolic file system links. Typically, they are used for short cuts in the NIS+ namespace.

For more information about the NIS+ objects, see `nis_objects`(3N).

NIS+ entry objects

The NIS+ tables consist of NIS+ entry objects. For each entry in the NIS+ table, an NIS+ entry object exists. NIS+ entry objects conform to the schema defined by the NIS+ table object.

Multiple Administrative Domains

NIS+ allows the creation of multiple domains, or subset of the enterprise network, that may be administered on an autonomous basis. As a corporation grows, or as its corresponding domain grows, authorized administrators can subdivide the domain into two or more hierarchical subdomains.

NIS+ allows for a primary copy of information to be stored on a master server, with zero or more slave servers storing replicas of the primary copy. Updates are made only to the master server, which propagates them to its slave servers. An NIS+ client can send look-up requests to any of the replicas and update requests only to the master server. This arrangement has two benefits: inconsistent updates between tables is avoided because only one master exists, and either a master or a slave server can act as a back-up server for look-up requests.

Each domain in an NIS+ network has its own master server and also may have many slave replicated servers. The overall reliability of the network is enhanced when multiple master servers are across the network, one for each domain, as opposed to one master domain for an NIS+ network. If a master server is down, only updates for its particular domain are disabled; updates to the rest of the network are not affected.

NIS+ Names

The NIS+ service defines two forms of names, simple names and indexed names. The service uses simple names to identify NIS+ objects contained within the NIS+ namespace. Indexed names are used to identify NIS+ entries contained within NIS+ tables. Entries within NIS+ tables also are returned to the caller as NIS+ objects of type entry. NIS+ objects are implemented as a union structure, which is described in the <`rpcsvc/nis.h`> file. The `nis_objects`(3N) man page describes the differences between the various types and the components of these objects.

Simple Names

Simple names are made up of a series of labels that are separated by the dot (`.`) symbol. Each label is composed of printable symbols from the ISO Latin 1 set. Each label can be of any nonzero length, provided that the fully qualified name is fewer than `NIS_MAXNAMELEN` octets, including the separating dots. For the actual value of `NIS_MAXNAMELEN`, see the <`rpcsvc/nis.h`> header file. You must use quotation marks for labels that contain special symbols. See the Grammar subsection.

The NIS+ namespace is organized as an individual rooted tree. Simple names identify nodes within this tree. These names are constructed such that the leftmost label in a name identifies the leaf node, and all of the labels to the right of the leaf identify that objects parent node. The parent node is referred to as the *leafs directory* (this is a naming directory and should not be confused with a file system directory).

For example, the name `example.simple.name` is a simple name that has three labels:

`example` The leaf node in this name

`simple` The directory of this leaf

`name` Simple name of the directory

The `nis_leaf_of`(3N) function returns the first label of a simple name. The `nis_domain_of`(3N) function returns the name of the directory that contains the leaf. Repeated use of these two functions can break a simple name into each of its label components.

The name dot (`.`) is reserved to name the global root of the namespace. For systems that are connected to the Internet, this global root will be served by a domain name service (DNS). When an NIS+ server is serving a root directory whose name is not dot (`.`), this directory is referred to as a *local root*. The root of the NIS+ namespace does not have to be the local root, and it ends in a trailing dot.

NIS+ names are said to be fully qualified when the name includes all of the labels that identify all of the directories, up to the global root. Names without the trailing dot are called *partially qualified*.

Indexed Names

Indexed names are compound names that are composed of a search criterion and a simple name. The search criterion component is used to select entries from a table; the simple name component is used to identify the NIS+ table that will be searched. The search criterion is a series of column names and their desired values enclosed in bracket (`[]`) symbols. These criteria take the following form:

`[column_name=`*value*`, column_name=`*value*`, . . . ]`

A search criterion is combined with a simple name to form an indexed name by concatenating the two parts, separated by a comma (`,`) symbol, as follows.

```
[ search-criterion ],table.directory.
```

When multiple column name/value pairs are present in the search criterion, only those entries in the table that have the appropriate value in all columns specified are returned. When no column name/value pairs are specified in the search criterion, all entries in the table are returned.

Grammar

The following text represents a context-free grammar that defines the set of legal NIS+ names. The terminals in this grammar are the following symbols:

- Dot (`.`)
- Open bracket (`[`)
- Close bracket (`]`)
- Comma (`,`)
- Equals (`=`)
- White space

Angle brackets (< and >), which delineate nonterminals, are not part of the grammar. The vertical bar (`|`) symbol is used to separate alternate productions, and it should be read as either this production or the following production:

name	`::= . |` <*simple name*> `|` <*indexed name*>
simple name	`::=` <*string*>`. |` <*string*>`.`<*simple name*>
indexed name	`::=` <*search criterion*>`,`<*simple name*>
search criterion	`::= [` <*attribute list*> `]`
attribute list	`::=` <*attribute*> `|` <*attribute*>`,`<*attribute list*>
attribute	`::=` <*string*> `=` <*string*>
string	`::= ISO Latin 1 character set`

The `/` symbol is not used. The initial character may not be a terminal character or the symbols at (`@`), plus (`+`), or hyphen(-).

Terminals that appear in strings must be quoted with double quotation marks (`"`). You may quote the `"` symbol by quoting it with itself (`""`).

Name Expansion

The NIS+ service accepts only fully qualified names. Because such names may be unwieldy, however, the NIS+ commands use a set of standard expansion rules that will try to fully qualify a partially qualified name. The NIS+ library function `nis_getnames`(3N) actually does this expansion. This function generates a list of names by using the default NIS+ directory search path or the `NIS_PATH` environment variable. The default NIS+ directory search path includes all of the names in its path. When the `EXPAND_NAME` flag is used, the `nis_lookup`(3N) and `nis_list`(3N) functions invoke `nis_getnames`(3N).

The `NIS_PATH` environment variable contains an ordered list of simple names. The names are separated by the `:` symbol. If any name in the list contains colons, you should quote the colon as described in the Grammar subsection. When the list is exhausted, the resolution function returns the error `NIS_NOTFOUND`. This may end up masking the fact that the name existed but a server for it was unreachable. If the name presented to the list or look-up interface is fully qualified, the `EXPAND_NAME` flag is ignored.

In the list of names from the NIS_PATH environment variable, the $ symbol is treated specially. Simple names that end with the $ have this symbol replaced by the default directory. For more information, see nis_local_directory(3N). Using the $ as a name in this list results in this name being replaced by the list of directories between the default directory and the global root that contain at least two labels.

An example of this expansion follows. If the default directory is a long name (such as some.long.domain.name.), and the NIS_PATH variable is set to fred.bar.:org_dir.$:$, this path is initially broken up into the following list:

1. fred.bar.
2. org_dir.$
3. $

The $ in the second component is replaced by the default directory. The $ in the third component is replaced with the names of the directories between the default directory and the global root that have at least two labels in them. The effective path value becomes:

1. fred.bar.
2. org_dir.some.long.domain.name.
3. some.long.domain.name.
4. long.domain.name.
5. domain.name.

Each of these simple names is appended to the partially qualified name that was passed to the nis_lookup(3N) or nis_list(3N) interface. Each is tried until NIS_SUCCESS is returned or the list is exhausted.

If the NIS_PATH variable is not set, the path $ is used.

The nis_getnames(3N) function may be called from user programs to generate the list of names that would be searched. You also can use the nisdefaults(8) program with the -s option to show the fully expanded path.

Concatenation Path

Usually, all of the entries for a certain type of information are stored within the table itself. At times, however, it is desirable for the table to point to other tables where entries can be found. For example, you may want to store all IP addresses in the host table for their own domain, and yet want to be able to resolve hosts in some other domain without explicitly specifying the new domain name. With a concatenation path, you can create a sort of flat namespace out of a hierarchical structure. You also can create a table with no entries and just point the hosts or any other table to its parent domain. With such a set up, you are moving the administrative burden of managing the tables to the parent domain. The concatenation path slows down the request response time because more tables and more servers are searched.

NIS+ provides a mechanism for concatenating different but related tables with a "NIS+ Concatenation Path." This path is set up at table creation time by using the `nistbladm`(8) command. You can specify more than one table to be concatenated, and they are searched in the given order. The NIS+ client libraries will not follow the concatenation path set in the other tables.

Namespaces

The NIS+ service defines two additional disjoint namespaces for its own use. These namespaces are the NIS+ Principal namespace and the NIS+ Group namespace. The names associated with the group and principal namespaces are syntactically identical to simple names. However, the information they represent cannot be obtained by directly presenting these names to the NIS+ interfaces. Special interfaces are defined to map these names into NIS+ names so that they may then be resolved.

Principal Names

NIS+ principal names uniquely identify users and machines that are making NIS+ requests. These names have the following form:

principal.domain

The *domain* is the fully qualified name of an NIS+ directory in which the specified principals credentials can be found. For more information on domains, see the Directories and Domains subsections. No leaf exists in the NIS+ namespace in the name, *principal*.

Credentials are used to map the identity of a host or user from one context such as a process UID into the NIS+ context. They are stored as records in an NIS+ table named `cred`. `cred` is always found in the `org_dir` subdirectory of the directory specified in the principal name.

You can express this mapping as a replacement function, as follows:

principal.domain `->[cname=`*principal.domain*`],cred.org_dir.domain`

This latter name is an NIS+ name that can be presented to the `nis_list`(3N) interface for resolution. To administer the NIS+ principal names, use the `nisaddcred`(8) command.

The `cred` table contains the following five columns:

- `cname`
- `auth_name`
- `auth_type`
- `public_data`
- `private_data`

One record in this table exists for each identity mapping for an NIS+ principal. The current service supports two such mappings:

`LOCAL` Maps from the UID of a given process to the NIS+ principal name associated with that UID. If no mapping exists, the name `nobody` is returned. When the effective UID of the process is 0 (for example, the privileged user), the NIS+ name associated with the host is returned. UIDs are sensitive to the context of the machine on which the process is executing.

`DES` Maps to and from a Secure RPC netname into an NIS+ principal name. Because netnames contain the notion of a domain, they span NIS+ directories.

The NIS+ client library function `nis_local_principal`(3N) uses the `cred.org_dir` table to map the UNIX notion of an identity, a process UID, into an NIS+ principal name. Shell programs can use the command `nisdefaults`(8) with the `-p` option to return this information.

To map from UIDs to an NIS+ principal name, construct a query in the following form:

`[auth_type=LOCAL, auth_name=`*uid*`],cred.org_dir.`*defaultdomain*`.`

This query returns a record that contains the NIS+ principal name associated with this UID in the machines default domain.

The NIS+ service uses DES mapping to map the names associated with Secure RPC requests into NIS+ principal names. RPC requests that use Secure RPC include the netname of the client making the request in the RPC header. This netname has the following form:

`unix.`*UID@domain*

The service constructs a query by using the following form:

`[auth_type=DES, auth_name=`*netname*`],cred.org_dir.`*domain.*

The domain part is extracted from the netname, rather than using the default domain. This query is used to look up the mapping of this *netname* into an NIS+ principal name in the domain in which it was created.

This mechanism of mapping UID and network names into an NIS+ principal name ensures that a client of the NIS+ service has only one principal name. This principal name is used as the basis for authorization, which is described as follows. All objects in the NIS+ namespace and all entries in NIS+ tables must have an owner specified for them. This owner field always contains an NIS+ principal name.

Group Names

Like NIS+ principal names, NIS+ group names take the form:

group_name.domain

All objects in the NIS+ namespace and all entries in NIS+ tables may optionally have a group owner specified for them. This group owner field, when filled in, always contains the fully qualified NIS+ group name.

The NIS+ client library defines several interfaces for dealing with NIS+ groups. For information on these interfaces, see the `nis_groups`(3N) man page. These interfaces internally map NIS+ group names into an NIS+ simple name that identifies the NIS+ group object associated with that group name. This mapping looks like the following:

group.domain `->` *group.groups_dir.domain*

This mapping eliminates collisions between NIS+ group names and NIS+ directory names. For example, without this mapping, a directory with the name `engineering.foo.com.` would make it impossible to have a group named `engineering.foo.com.`. This is due to the restriction that within the NIS+ namespace, a name unambiguously identifies one object. With this mapping, the NIS+ group name `engineering.foo.com.` maps to the NIS+ object name `engineering.groups_dir.foo.com.`.

The contents of a group object is a list of NIS+ principal names, and the names of other NIS+ groups. For a more complete description of their use, see `nis_groups`(3N).

Directories and Domains

Some directories within the NIS+ namespace are referred to as NIS+ Domains. Domains are those NIS+ directories that contain the `groups_dir` and `org_dir` subdirectories. The `org_dir` subdirectory should contain the table named `cred`. In particular, because of the way the group namespace and the principal namespace are implemented within the NIS+ namespace, NIS+ Group names and NIS+ Principal names always include the NIS+ domain name after their first label.

NIS+ Security

Unlike NIS, NIS+ defines a security model to control access to information managed by the service. The service defines access rights that are selectively granted to individual clients or groups of clients. Principal names and group names are used to define clients and groups of clients that may be granted or denied access to NIS+ information.

The security model also uses the notion of a class of principals called `nobody` that contains all clients, whether or not they have authenticated themselves to the service and the class world. The class world includes any client who has been authenticated.

Authorization

The NIS+ service defines the following four access rights that can be granted or denied to clients of the service:

- `read`
- `modify`
- `create`
- `destroy`

These rights are specified in the object structure at creation time and may be modified later by using the `nischmod`(8) command. Generally, the rights granted for an object apply only to that object. However, for purposes of authorization, rights granted to clients reading directory and table objects are granted to those clients for all of the objects contained by the parent object. This notion of containment is abstract. The objects do not actually contain other objects within them. Group objects do contain the list of principals within their definition.

Access rights are interpreted as follows:

`read` This right grants read access to an object. For directory and table objects, having read access on the parent object conveys read access to all of the objects that are direct children of a directory, or entries within a table.

`modify` This right grants modification access to an existing object. Read access is not required for modification. In many applications, however, you must read an object before modifying it. Such modify operations will fail unless you also grant read access.

`create` This right gives a client permission to create new objects where one had not previously existed. It is used only in conjunction with directory and table objects. Create access for a table allows a client to add additional entries to the table. Create access for a directory allows a client to add new objects to an NIS+ directory.

`destroy` This right gives a client permission to destroy or remove an existing object or entry. When a client tries to destroy an entry or object by removing it, the service first checks to see whether the table or directory containing that object grants the client destroy access. If it does, the operation proceeds, if the containing object does not grant this right, the object itself is checked to see whether it grants this right to the client. If the object grants the right, the operation proceeds; otherwise, the request is rejected.

Each of these rights may be granted to any one of four different categories, as follows:

`owner` A right may be granted to the owner of an object. The owner is the NIS+ principal identified in the owner field. To change the owner, use the `nischown`(8) command. If the owner does not have modification access rights to the object, the owner cannot change any access rights to the object, unless the owner has modification access rights to its parent object.

`group owner` A right may be granted to the group owner of an object. This grants the right to any principal that is identified as a member of the group associated with the object. To change the group owner, use the `nischgrp`(8) command. The object owner does not have to be a member of this group.

`world` A right may be granted to everyone in the world. This grants the right to all clients who have authenticated themselves with the service.

`nobody` A right may be granted to the `nobody` principal. This has the effect of granting the right to any client that makes a request of the service regardless of whether or not they are authenticated.

For bootstrapping reasons, directory objects that are NIS+ domains, the `org_dir` subdirectory, and the `cred` table within that subdirectory must have `read` access to the `nobody` principal. This makes navigation of the namespace possible when a client is in the process of locating its credentials. Granting this access does not allow the contents of other tables (such as the password table) within `org_dir` to be read.

Directory authorization

Additional capabilities are provided for granting access rights to clients for directories. These rights are contained within the object access rights (OAR) structure of the directory. This structure allows the NIS+ service to grant rights that are not granted by the directory object to be granted for objects contained by the directory of a specific type.

An example of this capability is a directory object that does not grant create access to all clients, but does grant create access in the OAR structure for group type objects to clients who are members of the NIS+ group associated with the directory. In this example, the only objects that could be created as children of the directory would have to be of the `type` group.

Another example is a directory object that grants `create` access only to the owner of the directory, and additionally grants `create` access through the OAR structure for objects' types (for example, table, link, group, and private) to any member of the directories group. This OAR structure allows complete `create` access to a group except for creating subdirectories. This also restricts the creation of new NIS+ domains, because creating a domain requires creating both a `groups_dir` and `org_dir` subdirectory.

Currently, no command-line interface exists to set or change the object access rights of the directory object.

Table authorization

As with directories, additional capabilities are provided for granting access to entries within tables. Rights granted to a client by the access rights field in a table object apply to the table object and all of the entry objects contained by that table. If an access right is not granted by the table object, it may be granted by an entry within the table. This holds for all rights except `create`.

For example, a table may not grant `read` access to a client performing a `nis_list`(3N) operation on the table. However, the access rights field of entries within that table may grant `read` access to the client. Access rights in an entry are granted to the owner and group owner of the entry and not the owner or group of the table. When the list operation is performed, all entries to which the client has `read` access are returned. Those entries that do not grant `read` access are not returned. If none of the entries that match the search criterion grant `read` access to the client making the request, no entries are returned and the result status contains the `NIS_NOTFOUND` error code.

Access rights that are granted by the *rights* field in an entry are granted for the entire entry. In the table object, however, an additional set of access rights is maintained for each column in the table. These rights apply to the equivalent column in the entry. The rights are used to grant access when neither the table nor the entry itself grants access. The access rights in a column specification apply to the owner and group owner of the entry, rather than the owner and group owner of the table object.

When a `read` operation is performed, if `read` access is not granted by the table and is not granted by the entry but is granted by the access rights in a column, that entry is returned with the correct values in all columns that are readable and the string `*NP*` in columns in which `read` access is not granted.

As an example, consider a client that has performed a list operation on a table that does not grant read access to that client. Each entry object that satisfied the search criterion specified by the client is examined to see whether it grants `read` access to the client. If it does, it is included in the returned result. If it does not, each column is checked to see whether it grants `read` access to the client. If any columns grant `read` access to the client, data in those columns is returned. Columns that do not grant `read` access have their contents replaced by the string `*NP*`. If none of the columns grant `read` access, then the entry is not returned.

FILES

All clients of the NIS+ service should include the `rpcsvc/nis.h` header file.

SEE ALSO

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

`nis_groups`(3N), `nis_local_names`(3N), `nis_names`(3N), `nis_objects`(3N), `nis_subr`(3N), `nis_tables`(3N) in *ONC+ Technology for the UNICOS Operating System*, Cray Research publication SG–2169

`newkey`(8), `nisaddcred`(8), `nischown`(8), `nisdefaults`(8), `nismatch`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

nisfiles – NIS+ database files and directory structure

IMPLEMENTATION

Cray Research systems licensed for ONC+™ and UNICOS 8.3 or later

DESCRIPTION

The Network Information Service Plus (NIS+) uses a memory based, replicated database. This database uses a set of files in the /etc/nis directory for checkpointing to stable storage and for maintaining a transaction log. The NIS+ server and client also use files in this directory to store binding and state information.

The NIS+ service implements an authentication and authorization system that is built upon Secure RPC. In this implementation, the service uses a table named cred.org_dir.domainname to store the public and private keys of principals that are authorized to access the NIS+ namespace. It stores group access information in the subdomain groups_dir.domainname as group objects. These two tables appear as files in the /etc/nis/hostname directory on the NIS+ server.

Unlike the previous versions of the network information service in NIS+, the information in the tables is initially loaded into the service from the ASCII files on the server and then updated using NIS+ utilities. For details, see the nistbladm(8) man page and the -D option description.

The following files are stored in the /etc/nis directory:

NIS_COLDSTART
: This file contains NIS+ directory objects that will be preloaded into the NIS+ cache at start-up time. This file usually is created at NIS+ installation time. For more information, see nisinit(8).

NIS_SHARED_DIRCACHE
: This file contains the current cache of NIS+ bindings being maintained by the cache manager. To view the contents, use the nisshowcache(8) command.

hostname.log
: This file contains a transaction log that is maintained by the NIS+ service. To view it, use the nislog(8) command. This file contains holes. Its apparent size may be a lot larger than its actual size. There is only one transaction log per server.

hostname.dict
: This file is a dictionary that the NIS+ database uses to locate its files. The default NIS+ database package creates the dictionary. The dictionary has no log file.

hostname
: This directory contains databases that the server uses.

hostname/root.object
: On root servers, this file contains a directory object that describes the root of the namespace.

`hostname/parent.object`
: On root servers, this file contains a directory object that describes the parent namespace. The `nisinit`(8) command creates this file. If this is an isolated namespace, this file is not created.

`hostname`/*table_name*
: For each table in the directory, there will be a file with the same name that stores the information about that table. If subdirectories are within this directory, the database for the table is stored in the file *table_name*.subdirectory.

`hostname`/*table_name*.log
: This file contains the database log for the table *table_name*. The log file maintains the state of individual transactions to each database. When a database has been checkpointed (that is, all changes have been made to the `hostname`/*table_name* stable storage), this log file will have a length of 0.

  Currently, NIS+ does not do checkpointing automatically. Administrators should execute the `nisping`(8) command with the `-C` option once a day to checkpoint the log file. To accomplish this, use either a `cron`(8) job or execute the command manually each time. For more information, see `nisping`(8).

`hostname.root_dir`
: On root servers, this file stores the database associated with the `root` directory. It is similar to other table databases. The corresponding log file is called `root_dir.log`.

`hostname/cred.org_dir`
: This table contains the credentials of principals in this NIS+ domain.

`hostname/groups_dir`
: This table contains the group authorization objects that NIS+ needs to authorize group access.

NOTES

Except for the `NIS_COLDSTART` and the `NIS_SHARED_DIRCACHE` file, no other files should be manipulated by commands such as `cp`(1), `mv`(1), or `rm`(1). Because the transaction log file keeps logs of all changes made, you must not manipulate the files independently.

SEE ALSO

`nis`(4)

`nis_db`(3N), `nis_objects`(3N) in *ONC+ Technology for the UNICOS Operating System*, Cray Research publication SG–2169

`niscat`(8), `nisinit`(8), `nislog`(8), `nismatch`(8), `nisping`(8), `nistbladm`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

np – Network packet driver for low-speed interfaces

IMPLEMENTATION

Cray PVP systems with an IOS model E

DESCRIPTION

The np driver provides an interface for all of the low-speed network devices connected to an IOS model E (IOS-E), including NSC devices such as the N130 and the FEI3 VME interface. The driver accepts standard UNICOS read(2), write(2), open(2), close(2), listio(2), reada(2), and writea(2) system calls; it also accepts ioctl(2) requests.

By convention, the N-packet special file names are in the following format:

/dev/comm/*ioc iop chan*/lp *nn*

ioc Single-digit I/O cluster number.

iop Single-digit IOP number.

chan Two-digit octal channel number.

nn Low-order 4 bits of the minor device number ("logical path" in IOS terminology).

Usually, logical path 5 (/dev/comm/*/lp05) is reserved for TCP/IP. You also may use other minor devices, depending on how the interfaces are configured. Check with your system support staff for details of usage at your site. The contents of the device node determine the configuration of the communication device; see the Configuration subsection.

You cannot open a minor device when it is already open. Any attempt to do so fails with an EBUSY error. Each device has a maximum of 16 paths. For driver type NP_FLG_RAW, you may open only one path.

ioctl Requests

The ioctl(2) requests described in this section are defined in sys/np.h.

The following ioctl(2) request requires no parameters:

NPC_HLTIO Halts an outstanding I/O request. This is valid only for asynchronous I/O. All synchronous requests will block.

The following ioctl(2) request requires one parameter in the npioctl structure:

NPC_DEVCNTL Network interface device control. The valid values for sfunc are as follows (see epackn.h):

NP_DC_MC	0	Channel master clear
NP_DC_OD	1	Output disconnect
NP_DC_AUTO_OD	2	Set auto disconnect mode in CCA1
NP_DC_CLR_AUTO_OD	3	Clear auto disconnect mode in CCA1

The following `ioctl`(2) requests require parameters in the `npstat` structure. The `npstat` structure is as follows:

```
struct npstat {
        int     sfunc;          /* status request subfunction            */
        char    *sbuf;          /* status buffer pointer                 */
        int     dev;            /* device number (-1 means all)          */
        int     lpath;          /* logical path number (-1 means all)    */
        uint    slen;           /* status buffer length (bytes)          */
        uint    epoch;          /* incremented every configuration change */
};
```

`NPC_CDSTATS` Clears device statistics request. This is a super-user-only request. The user passes the address of an `npstat` structure by using the `ioctl`(2) request. The requested (all, if `dev` = `-1`) device statistics are cleared. The `epoch` variable is returned in the `npstat` structure.

`NPC_CLSTATS` Clears logical path statistics request. This is a super-user-only request. The user passes the address of an `npstat` structure by using the `ioctl`(2) request. The requested (all, if `lpath` = `-1`) path statistics are cleared. The `epoch` variable is returned in the `npstat` structure.

`NPC_DEV_STATUS` Network logical channel statistics request. This is a super-user-only request. The channel statistics are placed in a buffer specified in the request. The structures that follow define the statistics.

The first five words of `NPC_DEV_STATUS` are status from the configure up request. The first word is channel status:

```
uint    nsr_ibz:  1,    /* input channel busy flag  */
        nsr_idn:  1,    /* input channel done flag  */
                 14,
        nsr_ics: 16,    /* input channel status     */
        nsr_obz:  1,    /* output channel busy flag */
        nsr_odn:  1,    /* output channel done flag */
                 14,
        nsr_ocs: 16;    /* output channel status    */
```

The next four words of `NPC_DEV_STATUS` are valid only for N130 devices. If the master clear request was successful, the response is the "Initialize Device Response Parameter Block" that the device sends to the IOS; otherwise, the response is the response to the failing master clear request.

Each open logical path has the last read/write reply trailer from the IOS added to the buffer. The path is identified by one integer that contains the path number. The next word contains the last error from the IOS for that path. The read/write reply trailer is defined as follows:

```
struct np_rw_rep {                      /* read/write reply trailer     */
        uint    nprw_bz  :1,            /* busy flag                    */
                nprw_dn  :1,            /* done flag                    */
                         :14,           /* unused                       */
                nprw_mdc :16,           /* messages discarded (channel) */
                nprw_mdp :16,           /* messages discarded (lchan)   */
                nprw_mbp :16;           /* messages buffered on lchan   */
        uint             :32,           /* unused                       */
                nprw_iob_addr :32;      /* I/O buffer address           */
};

struct np_rw_reps {     /* read/write reply trailer stats  (optional) */
        uint    nprw_status[4];      /* device status                 */
};
```

These structures are replicated for read and write for each logical path:

```
struct np_rw_stats {
        struct np_rw_rep np_rdstats;    /* read statistics              */
        struct np_rw_reps np_rd_dstats; /* read device statistics       */
        struct np_rw_rep np_wrstats;    /* write statistics             */
        struct np_rw_reps np_wr_dstats; /* write device statistics      */
};
```

The entire structure is defined in `sys/np.h`. The supporting structures are in `epackn.h` and `npsys.h`.

`NPC_DSTATUS` Network interface status request. This is a super-user-only request. This request returns statistics from all channel-related activity on the IOS. The user passes the address of an `npstat` structure by using the `ioctl`(2) request. The length of data and the `epoch` variable are returned in the `npstat` structure.

`NPC_ECHO` N-packet echo. This `ioctl`(2) request lets a super user send echo packets to a communications driver in an IOP. The IOP returns the packets; this function may be used to verify and time the request/response path of an N-packet through the IOS.

`NPC_LSTAT` Last status of interface. This request returns the last status from the IOS. If a request returns an error to the user, the `errno` is a generic EIO. This `ioctl`(2) request lets users determine the exact cause of the failure. The errors are defined in `sys/epackn.h`. Recovery from error cases depends on the type of error and the type of interface. Before implementing any recovery techniques, you should thoroughly understand the device and error modes.

`NPC_LSTATUS` Network logical path status request. This is a super-user-only request. This request returns statistics from all path-related activity on the IOS. The user passes the address of an `npstat` structure by using the `ioctl`(2) request. The length of data and the `epoch` variable are returned in the `npstat` structure.

`NPC_PACKET` N-packet interface to allow a super user to issue N-packets directly from a user program. This request gives the user complete control of the IOP interface to a specific device.

`NPC_STAT` Network interface status request. This is a super-user-only request. This request returns statistics from all channel-related activity on the IOS. This is deferred.

Several "driver types" are defined in the configuration. The raw driver allows a process to send data in any format. The other drivers require a network header to be the first words of any data. This header is an NSC message proper (for types MP, PB, and A130).

The message proper is defined as follows:

```
struct mp {
        char  control[2];       /* NSC control word        */
        char  acode[2];         /* NSC access code         */
        char  to[2];            /* NSC destination adapter */
        char  from[2];          /* NSC source adapter      */
        char  param[56];        /* NSC parameters          */
     } ;
```

The `mp` structure is defined in `sys/np.h`. The details of the contents of message proper fields are available in NSC documentation. For most devices such as the VME interfaces, the important fields are the *to* and *from* fields. You can determine the contents of these fields from the network "adapter" addresses of the host computers and the logical path (device minor).

If a `read`(2) system call is not satisfied within the time-out period, an `ELATE` error is returned. If a `write`(2) system call cannot be completed, it may be retried periodically in the time allowed by the time-out period before an `ELATE` error is returned. A `close`(2) system call closes the minor device. Any outstanding system calls are terminated with an EIO error. The time-outs are defined in the N-packet include files, which you should not have to change.

Configuration

The following definitions are used for the `mknod`(8) parameters:

`/etc/mknod` *name* `c` *maj min ioc iop chan cmode dtype dmode adap hwtype*

name Name of the special file, usually `/dev/comm/`*ioc iop chan* `/lp` *nn*

maj Major device number, always 35 for the `np.c` driver

min Minor device number (encoded device and logical path)

ioc IOS cluster number [0-7] [00-07]

iop IOS processor number [0, 1, 2, or 3]

chan IOP channel number in octal [030, 032, 034, or 036]

cmode Controller mode [0: 6 Mbyte; 1: 12 Mbyte; 2: 12 Mbyte loopback]

dtype Driver type (see below for *dmode*; meaning depends on `dtype`)

0 Raw driver: *dmode* not used

1 Message proper (FEI-3, Cray-Cray): *dmode* = 0
Message proper (VAXBI): *dmode* = 1

2 Parameter block driver (NSC N130, Ultra LSC)

The following *dmode* bits are valid only for the N130:

dmode bits 0-15 = 0: no special functions in N130
dmode bits 16-31 (`dfunc`) = 0: no driver function in N130
dmode bit 8 - 0400: variable length message propers (mp's) on medium
dmode bit 7 - 0200: CRC (deferred)
dmode bit 6 - 0100: statistics on in N130
dmode bit 5 - 0040: adapter microcode trace on in N130
dmode bit 4 - 0020: send disconnect after parameter block (N130)
dmode bit 3 - 0010: disable write response parameter block (N130)
dmode bit 2 - 0004: DXU master clear at power up (N130)
dmode bit 1 - 0002: purge all network data (N130)
dmode bit 0 - 0001: clear interface only (N130)

4 NSC message proper (A130, CNT LANlord) : *dmode* n/u

dmode Driver function and driver mode (high-order 16 bits is driver function). Currently, only the PB driver uses the driver function to specify microcode trace modes.

adap A130/N130 adapter address (in hexadecimal)

hwtype Code that specifies the type of hardware or adapter on the channel; for a detailed list of codes, see `netdev.h`. Only monitoring software uses this code.

0102 FEI-3

0104 FEI-CN

0105 FEI-DS

0106 FEI-UC

0107 FEI-VA

0110 FEI-VB

0111 FEI-VM

0301 A130 HYPERchannel

0302	N130 HYPERchannel
0303	EN643 Ethernet
0304	DX4130 FDDI
0401	VAXBI
0502	Ultra LSC

NOTES

A flag is required in the inode for control devices for monitors, configuration commands, and so on. This flag disables input, output, and the process of incrementing the `epoch` variable when the device is opened and closed.

WARNINGS

Differences exist between the interfaces supported by this driver and those supported by previous IOS drivers. Do not use include files from `hy`(4) with this interface.

You can open only one logical path for type RAW.

MESSAGES

The driver returns the following error codes:

`EBUSY` The device special file is currently in use.
The device is type RAW, and another path is in use.

`EFAULT` An `ioctl`(2) request did not have enough buffer space for the data returned.

`ENOTTY` An attempt was made to close a logical path that was not open.

`ENXIO` Device special file has a bad channel number.
Device special file has a bad minor number.
Network device structures are all in use.
Network logical path structures are all in use.
Device special file has 6-Mbyte set for an N130 device.
Attempt was made to close a device that was not open.
Attempt was made to close a logical path that was not allocated.
Attempt was made to execute an `ioctl`(2) command to issue an device that was not open.
Attempt was made to perform an unknown `ioctl` code. `NPC_DEVCNTL ioctl`(2) request had a bad function code or was not from a super user.
`NPC_ECHO ioctl`(2) request was not from a super user.
`NPC_PACKET ioctl`(2) request was not from a super user.

The driver writes the following error messages to the system log:

```
ERROR: np.c: Cannot allocate device structure
ERROR: np.c: Cannot allocate logical channel
WARNING: np.c: npstrat: unknown driver type
WARNING: np.c: N-packet structure in use.
WARNING: np.c: npintr: no bp structure.
INFO: np.c: path closed - packet returned
INFO: np.c: No free logical devices
INFO: np.c: No free device structures
np.c: output sequence error %x %x
np ioc %d IOP %d ch 0%o error %d (0%o)
np.c: receive pkt sequence error %x %x
np.c: np device structure closed - reopening
npl already in use
```

For IOS-reported errors, the following message is logged:

```
np ioc %d IOP %d ch 0%o error %d (0%o)
```

The driver returns the following error codes:

Value	**Definition**
10	Device-detected error
11	Parity error
12	SECDED error
20	Retry of failing request unsuccessful

The following are software detected errors; execution attempted and failed:

Value	**Definition**
100	Local memory not available.
101	IOB memory not available.
102	Driver terminated.
103	Overrun on read request of N-packet; returned data is truncated.
104	CCA-1 hardware information is not valid.
105	CCA-1 input channel time-out.
106	CCA-1 output channel time-out.
107	Halt I/O request.
110	Maximum consecutive errors encountered; driver terminated.
111	Transferred fewer parcels than requested.
112	Cannot create a required IOS-E activity.

113	A parameter block that was not valid was received off of the CCA-a input channel
114	Read request packet time-out.
115	Cannot drain input channel completely at initialization.
116	Cannot buffer entire input; input truncated.
117	Request aborted due to `CLOSE PATH` request.
120	`RELMEM` request failed.
121	Cannot terminate (`TERM`) all driver activities as desired.
122	Microcode in device not supported.
123	Cannot halt I/O in a driver activity as desired.
250	Bad parameter on `TIMER` call.
251	Attempt to start `TIMER` that is already active.
252	Attempt to stop a `TIMER` that is not active.
260	Target memory I/O error: bad channel buffer ordinal.
261	Target memory I/O error: bad transfer direction.
262	Target memory I/O error: bad channel buffer address.
263	Target memory I/O error: hardware error on HISP channel.
264	Target memory I/O error: target memory not available.
265	Target memory I/O error: bad word length parameter.
270	Local memory to or from channel buffer error: bad buffer ordinal.
271	Local memory to or from channel buffer error: bad transfer direction.
272	Local memory to or from channel buffer error: bad buffer address.
273	Local memory to or from channel buffer error: hardware error on I/O.
274	Local memory to or from channel buffer error: bad word length parameter.
277	IOS-E internal error.

The following are parameter errors in the request packet; execution not tried:

Value	**Definition**
300	Packet type is not valid
301	Request code is not valid
302	Channel not configured up
303	Channel number is not valid
304	Channel already configured up

305	No connection path open for this logical path
306	Logical path is not valid
307	Logical path already open
310	CCA-1 mode is not valid
311	Driver type is not valid
312	Driver mode is not valid
313	Requested transfer length is not valid
314	Subfunction is not valid
315	Requested information not available
316	Packet length is not valid
317	Time-out value is not valid (must be nonzero)
320	Initialization of channel pair already in progress
321	Termination of channel pair already in progress
322	Second `OPEN PATH` request on CCA-1-Raw channel
323	Combination of driver type and driver mode is not valid
324	A130 driver not available in hybrid system
325	PB driver mode "input disc after PB" not available
377	Bad packet type (issued by monitor only)

EXAMPLES

This example makes the devices for the following configuration:

device 0: FEI-3 on cluster 3, IOP 1, channel 030, 6-Mbyte mode, logical paths 0 through 7
device 1: N130 on cluster 0, IOP 0, channel 032, (12-Mbyte mode), logical paths 0 through 7
device 2: FEI-1 on cluster 1, IOP 0, channel 036, 6-Mbyte mode, IBM MVS
device 3: FEI-1 on cluster 2, IOP 1, channel 034, 6-Mbyte mode, VAX on port A

```
/etc/mkdir /dev/comm
cd /dev/comm
/etc/mkdir v31-30 n00-32 f10-36 f21-34
cd v31-30
/etc/mknod lp00 c 35 000 3 1 030 0 1 0 0 0102
/etc/mknod lp01 c 35 001 3 1 030 0 1 0 0 0102
/etc/mknod lp02 c 35 002 3 1 030 0 1 0 0 0102
/etc/mknod lp03 c 35 003 3 1 030 0 1 0 0 0102
/etc/mknod lp04 c 35 004 3 1 030 0 1 0 0 0102
/etc/mknod lp05 c 35 005 3 1 030 0 1 0 0 0102
/etc/mknod lp06 c 35 006 3 1 030 0 1 0 0 0102
/etc/mknod lp07 c 35 007 3 1 030 0 1 0 0 0102
cd ../n00-32
/etc/mknod lp00 c 35 020 0 0 032 1 2 0 0 0302
/etc/mknod lp01 c 35 021 0 0 032 1 2 0 0 0302
/etc/mknod lp02 c 35 022 0 0 032 1 2 0 0 0302
/etc/mknod lp03 c 35 023 0 0 032 1 2 0 0 0302
/etc/mknod lp04 c 35 024 0 0 032 1 2 0 0 0302
/etc/mknod lp05 c 35 025 0 0 032 1 2 0 0 0302
/etc/mknod lp06 c 35 026 0 0 032 1 2 0 0 0302
/etc/mknod lp07 c 35 027 0 0 032 1 2 0 0 0302
cd ../f10-36
/etc/mknod lp00 c 35 040 1 0 036 0 3 0 0 0
cd ../f21-34
/etc/mknod lp00 c 35 060 2 1 034 0 3 1 0 0107
```

TCP/IP Configuration Example

Configuration for TCP/IP is done as described previously, except that the device names are `/dev/comm/tcp` *nnnn*; *nnnn* is an octal number, as follows:

The low-order 4 bits of the minor device number are the logical path. The high-order bits are the TCP/IP device number. Therefore, the following is true:

np0 has minors 0 through 17
np1 has minors 20 through 37
np2 has minors 40 through 57
np3 has minors 60 through 77
np4 has minors 100 through 117
np5 has minors 120 through 137
np6 has minors 140 through 157
np7 has minors 160 through 177, and so on

The next example makes the TCP/IP devices for the following configuration:

np0: FEI-3 on cluster 0, IOP 0, channel 030, 6-Mbyte mode, logical path 5
np1: N130 on cluster 0, IOP 0, channel 032, 12-Mbyte mode, logical path 5
np2: Cray channel on cluster 1, IOP 0, channel 036, 6-Mbyte mode, logical path 3
np3: VAXBI on cluster 2, IOP 1, channel 034, 12-Mbyte mode, logical path 5

```
/etc/mkdir /dev/comm
cd /dev/comm
/etc/mknod tcp0005 c 35 005
/etc/mknod tcp0025 c 35 025
/etc/mknod tcp0043 c 35 043
/etc/mknod tcp0065 c 35 065
```

FILES

`/dev/comm/*`	Device special files
`/usr/include/sys/np.h`	Definitions of constants and structures
`/usr/include/sys/npsys.h`	Definitions of constants and structures

SEE ALSO

fei(4), hy(4), vme(4)

ioctl(2), listio(2), read(2), reada(2), write(2), writea(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

NAME

npcntl – N-packet control interface

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

The N-packet control driver provides an interface for controlling the on/off status of NSC HYPERchannel adapters connected to the I/O subsystem (IOS). The N-packet control driver also is used for Cray Research front-end interfaces (FEIs) and VME interfaces attached to the IOS. For more information, see fei(4) and vme(4). The driver accepts standard UNICOS open(2) and close(2) system calls; it also accepts ioctl(2) requests.

The N-packet control driver is represented by the /dev/npcntl special file. Only the super user can use the device, because turning the network channels on and off interrupts network traffic.

You can use the ioctl request NPFC_CONFCHN to change an N-packet channel status in the IOS. The ioctl structure is defined in the sys/npcntl.h include file.

```
struct npc_cntrl {
        int     channel;  /* channel to change */
        int     ios;      /* ios to change     */
        int     state;    /* on/off status     */
        int     mode;     /* channel mode      */
};
```

channel The N-packet channel that is changing state.

ios The IOS to which the channel is connected.

state The state (either 0 (off) or 1 (on)) to which the channel is changing.

mode The mode in which to initiate the channel. The default mode is NSC; all other modes are deferred.

FILES

/dev/npcntl

/usr/include/sys/npcntl.h

SEE ALSO

fei(4), hy(4), vme(4)

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

NAME

nsswitch – Configuration file for the name-service switch

IMPLEMENTATION

Cray Research systems licensed for ONC+™ and UNICOS 8.3 or later

DESCRIPTION

The operating system uses many databases of information about users, groups, and so forth. Data for some of these databases come from a variety of sources. These sources and their lookup-order can be specified in the `/etc/nsswitch.conf` file.

The following databases use the switch:

Database	**Used By**
`automount`	`automount`(8)
`group`	`getgrent`(3C)
`passwd`	`getpwent`(3C)
`protocols`	`getprotobyname`(3C)
`publickey`	`getpublickey`(3R)
`rpc`	`getrpcbyname`(3C)
`services`	`getservbyname`(3C)
`netgroup`	`netgroup`(5)

You may use the following sources:

Source	**Uses**
`files`	`/etc/passwd`
`nis`	NIS (YP)
`nisplus`	NIS+

An entry in `/etc/nsswitch.conf` exists for each database. Typically, these entries will be simple, such as the following:

`protocols: files` or `networks: files nisplus`.

When you specify multiple sources, you may have to define precisely the circumstances under which each source will be tried.

A source returns one of the following status codes:

Status	**Meaning**
SUCCESS	Requested database entry was found
UNAVAIL	Source is not responding or corrupted
NOTFOUND	Source responded no such entry
TRYAGAIN	Source is busy, might respond to retries

For each status code, two actions are possible:

Action	**Meaning**
continue	Try the next source in the list
return	Return now

The complete syntax of an entry follows:

<*entry*>	k::= <*database*> : [<*source*> [<*criteria*>]]* <*source*>
<*criteria*>	::= [<*criterion*>+]
<*criterion*>	::= <*status*> = <*action*>
<*status*>	::=success | notfound | unavail | tryagain
<*action*>	::= return | continue

Each entry occupies one line in the file. Lines that are blank or that start with # symbol or with white space are ignored. The <*database*> and <*source*> names are case-sensitive, but <*action*> and <*status*> names are case-insensitive.

The library routines contain default entries that are used if the appropriate entry in `nsswitch.conf` is absent or syntactically incorrect.

The default criteria is to continue on anything except SUCCESS; that is, [SUCCESS=return NOTFOUND=continue UNAVAIL=continue TRYAGAIN=continue].

The default, or explicitly specified, criteria is meaningless following the last source in an entry, and it is ignored because the action is always to return to the caller regardless of the status code that the source returns.

Interaction with NIS+ and YP-compatibility Mode

The NIS+ server can be run in YP-compatibility mode. When you specify this mode, the server handles NIS (YP) requests and NIS+ requests. The results are the same, except that the `getpwent`(3C) routine uses the `nis` source rather than `nisplus`. You should use the `nisplus` source rather than the `nis` source.

Useful Configurations

The default entries for all databases use NIS+ as the enterprise level name-service. They are identical to those in the default configuration of this file:

Category	Entry
`passwd:`	`files nisplus`
`group:`	`files nisplus`
`protocols:`	`nisplus [NOTFOUND=return] files`
`rpc:`	`nisplus [NOTFOUND=return] files`
`ethers:`	`nisplus [NOTFOUND=return] files`
`publickey:`	`nisplus [NOTFOUND=return] files`
`automount:`	`files nisplus`
`services:`	`nisplus [NOTFOUND=return] files`

The policy `nisplus [NOTFOUND=return] files` implies that if `nisplus` is unavailable, continue on to `files`; if `nisplus` returns `NOTFOUND`, return to the caller. That is, treat `nis` as the authoritative source of information and try `files` only if `nisplus` is down.

NOTES

Within each process that uses `nsswitch.conf`, the entire file is read only once; if the file is changed later, the process will continue using the old configuration.

You should not use both `nis` and `nisplus` as sources for the same database because both name services are expected to store similar information and the lookups on the database may yield different results, depending on which name-service is operational at the time of the request.

Misspelled names of sources and databases will be treated as legitimate names of nonexistent sources and databases.

FILES

`/etc/nsswitch.conf`	Configuration file
`/etc/nsswitch.files`	Sample configuration file that uses only `files`
`/etc/nsswitch.nis`	Sample configuration file that uses `files` and `nis`
`/etc/nsswitch.nisplus`	Sample configuration file that uses `files` and `nisplus`

SEE ALSO

`netconfig`(4), `nis`(4), `ypfiles`(5) in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

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

NAME

`null` – Null file

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `/dev/null` file is a character special file. Data written on `/dev/null` is discarded; read operations from `/dev/null` always return 0 bytes.

FILES

`/dev/MAKE.DEV`

`/dev/null`

SEE ALSO

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

NAME

pdd – Physical disk device interface

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

The files in /dev/pdd are special files that allow read and write operations to physical disk devices. Each file represents one slice of a physical disk device. The files in /dev/pdd are character special files that may be used directly to read and write physical disk slices. Usually, they are called to perform I/O on behalf of higher-level logical disk device drivers. For I/O on a character disk device, read and write operations must transfer multiples of the physical device sector size and all seek operations must be on physical sector size boundaries.

The files in /dev/pdd are not usually mountable as file systems, although you may combine one or more physical disk slices to make a mountable logical disk device (see dsk(4), ldd(4), and mount(8)).

The files in /dev/pdd are created by using the mknod command (see mknod(8)). Each must have a unique minor device number, along with other parameters used to define a physical disk slice.

The mknod(8) command for physical disk devices is as follows:

mknod *name type major minor dtype iopath start length flags altpath unit*

name Descriptive file name for the device (for example, pdd/scr0230).

type Type of the device data being transferred. Devices in /dev/pdd are character devices denoted by a c.

major Major device number for physical disk devices. The dev_pdd name label in the /usr/src/uts/c1/cf/devsw.c file denotes the major device number for physical disk devices. You can specify the major number as dev_pdd.

minor Minor device number for this slice.

dtype Physical disk device types are defined in /usr/src/uts/c1/sys/pddtypes.h. Supported physical disk device types are as follows:

```
#define DD49    3       /* DD49 disk drive  */
#define DD40    6       /* DD40 disk drive  */
#define DD50    7       /* DD50 disk drive  */
#define DD41    9       /* DD41 disk drive  */
#define DD60    10      /* DD60 disk drive  */
#define DD61    11      /* DD61 disk drive  */
#define DD62    12      /* DD62 disk drive  */
#define DD42    13      /* DD42 disk drive  */
#define DA62    14      /* DA62 disk drive  */
#define DA60    15      /* DA60 disk drive  */
#define DD301   16      /* DD301 disk drive */
#define DA301   17      /* DA301 disk drive */
#define DD302   18      /* DD302 disk drive */
#define DA302   19      /* DA302 disk drive */
```

CRAY EL series disk types:

```
#define DDESDI  64      /* old esdi drive */
#define DD3     65      /* new esdi drive */
#define DDLDAS  66      /* old Max Strat DAS */
#define DDAS2   67      /* new Max Strat DAS */
#define DD4     68      /* ipi + sabre 7 */
#define RD1     69      /* removable esdi */
#define DDIMEM  70      /* ironics+memory vme boards */
#define DD5S    71      /* DD5S SCSI drive */
#define DD5I    72      /* DD5I IPI drive */
```

iopath The *iopath* specifies the I/O cluster, the I/O processor (IOP), and the controller channel number. For example, an *iopath* of 01234 is IOC 1, IOP 2, channel 34. The *iopath*, defined by the `io_path` structure in `sys/pdd.h`, follows. The structure is different for CRAY EL series, CRAY J90 series, and Cray PVP systems with an IOS model E. The unit is not used here; it is in a separate field, described below.

```
/*
 *  i/o path to the channel adapter
 */
struct io_path {
#if defined(CRAYEL)
        uint            :32,            /* must remain unused */
                unit    :8,
                ioc     :8,             /* ios - vme backplane */
                iop     :8,             /* eiop - controller   */
                chan    :8;             /* channel             */
#elif defined(CRAYJ90)
        uint            :32,            /* must remain unused */
                unit    :8,
                ioc     :6,             /* ios - vme backplane */
                iop     :9,             /* eiop - controller   */
                chan    :9;             /* channel             */
#else
        uint            :32,            /* unused */
                unit    :16,            /* unit */
                        :3,             /* unused */
                ioc     :4,             /* io cluster */
                iop     :3,             /* io processor */
                chan    :6;             /* channel */
#endif /* CRAYELS */
};
```

start Absolute starting block (sector) number of the slice.

length Number of blocks (sectors) in the slice.

flags Flags for physical disk device control, defined in `sys/eslice.h`, follows. They are mainly used for diagnostic and maintenance purposes. Usually, the flags field should be 0 for slices in `/dev/pdd`.

```
#define S_CONTROL       001    /* control device              */
#define S_NOBBF         002    /* no bad block forwarding     */
#define S_NOERREC       004    /* no error recovery           */
#define S_NOLOG         010    /* no error logging            */
#define S_NOWRITEB      020    /* no write behind             */
#define S_CWE           040    /* control device write enable */
#define S_NOSPIRAL      0100   /* no spiraling                */
```

altpath The optional alternate *iopath* that you can use as a back-up path to the physical disk device’s second port.

unit The disk device unit number for device types that support multiple units on the same channel.

`ioctl` Requests

The physical disk driver supports the following `ioctl`(2) requests. They are defined in `sys/pddtypes.h`, and they are passed as the *cmd* argument in the `ioctl`(2) system call.

If the `ioctl` description indicates that the `ioctl`(2) request has no effect on CRAY EL series systems, the call may be part of a command that the system supports, but the call has no meaning on the system. If the description indicates that the `ioctl`(2) request is not supported on CRAY EL series systems, the request is used only by commands not supported on CRAY EL series systems.

`PDI_STOP`	(Not supported on the CRAY EL series) Stops the queued disk requests after all outstanding requests finish.
`PDI_START`	(Not supported on the CRAY EL series) Resumes disk requests after they are stopped by using a `PDI_STOP ioctl`(2) request.
`PDI_DOWN`	Puts device in a down state and terminates all queued requests with an error.
`PDI_UP`	Puts device in an up state.
`PDI_RDONLY`	Sets device to a read-only state.
`PDI_NOALLOC`	Sets device to a state in which writes can occur but no new file allocation can take place. The file system uses this request.
`PDI_SPINIT`	(Not supported on the CRAY EL series) Initializes the spare sector map for the specified device.
`PDI_DIAG_REQ`	(Not supported on the CRAY EL series) Registers the calling process for a diagnostic function request. A `read`(2) or a `write`(2) system call by the calling process at a later time is treated as a diagnostic request. The argument is a pointer to a disk request packet, `drq_pak`, defined in `sys/epackd.h`.
`PDI_DIAG_RES`	(Not supported on the CRAY EL series) Registers the calling process for a diagnostic function response. An IOS response to a `read`(2) or a `write`(2) system call by the calling process at a later time is copied into the caller. The argument is a pointer to a disk response packet to which the response is copied. The disk response packet, `drs_pak`, is defined in `sys/epackd.h`.
`PDI_GETFLAGS`	(Not supported on the CRAY EL series) Copies the physical device control flags to the word to which *arg* points. The device control flags are defined previously.
`PDI_SETFLAGS`	(Not supported on the CRAY EL series) Sets the physical device control flags to the contents of *arg*. The device control flags are defined previously.
`PDI_SPIN_UP`	(Has no effect on the CRAY EL series) Issues a spin-up function to the physical device. The device must have this capability and be in remote mode.
`PDI_SPIN_DOWN`	(Has no effect on the CRAY EL series) Issues a spin-down function to the physical device. The device must have this capability and be in remote mode.
`PDI_GETMODE`	Gets the current read/write mode.

PDI_GETSTATE	Gets the current disk state.
PDI_PRIMARY	Selects primary path to disk.
PDI_ALTERNATE	(Has no effect on the CRAY EL series) Selects alternate path to disk.
PDI_RESET	Resets device stats.
PDI_GET_STREAMS	(Has no effect on the CRAY EL series) Gets streams.
PDI_SET_STREAMS	(Has no effect on the CRAY EL series) Sets streams.
PDI_GET_SL_STREAMS	(Has no effect on the CRAY EL series) Gets slice streams.
PDI_SET_SL_STREAMS	(Has no effect on the CRAY EL series) Sets slice streams.
PDI_ATOM_CP	(Not supported on the CRAY EL series) Atomic read/write diagnostic function.
PDI_RESYNC	(Has no effect on the CRAY EL series) Resyncs labels to spindle within array.
PDI_LDFRMT	(Not supported on the CRAY EL series) Loads format spec to spindle within array.

EXAMPLES

The following mknod(8) command makes a node for pdd/scr0230, type c, major number dev_pdd, minor number 110, disk type DD-60, I/O cluster 0, IOP 2, channel 30, startg at block 0, length of 1472 blocks, 0 for flags, no alternate path, and unit number of 1:

```
mknod pdd/scr0230 c dev_pdd 110 10 0230 0 1472 0 0 1
```

FILES

/dev/pdd/*

/usr/include/sys/pdd.h

/usr/include/sys/pddprof.h

/usr/src/c1/io/pdd.c

SEE ALSO

dsk(4), ldd(4), mdd(4), sdd(4), ssdd(4)

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

NAME

`proc` – Process file system

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `/proc` file system allows users access to the address space of a running process. This file system consists of files named `/proc/`*nnnnn*; *nnnnn* is the process ID formatted in decimal. Each file contains the address space of the process it represents.

Access to each member of `/proc` is restricted by the typical file system protection mechanisms, with the additional restrictions that operations such as `chown`(1) and `chmod`(1) are prohibited in the `/proc` file system. The `/proc` file system does not have an associated device driver, but a major device number is still needed for its current operation.

To inspect or modify the address space of a process by using the `/proc` file system, open the file in `/proc` that represents that process by using the `open`(2) system call, and then use the `lseek`(2), `read`(2), or `write`(2) system call to access the process's address space.

Additional operations on processes opened through `/proc` are supported by the `ioctl`(2) system call, allowing debuggers to control the execution of the subject process precisely and to obtain a file descriptor that refers to the subject process's text file. The `/proc ioctl` Requests subsection describes all `/proc ioctl`(2) operations.

Configuration

To create a `/proc` file system, use the following steps:

1. Create an empty directory called `/proc` by using the following command:

   ```
   mkdir /proc
   ```

2. Modify the system mount scripts and `/etc/fstab` file so that the `/proc` file system is always mounted at system startup. For the format of the `/proc` entry in the `fstab` file, see `fstab`(5).

Process Address Space Segmentation

When a process is opened through the `/proc` file system, its address space is segmented into several distinct address spaces. These separate process address spaces are declared in the `sys/fs/prfcntl.h` include file and are defined as follows:

`PRFS_DATA` Program data space.

`PRFS_TEXT` Program text space.

`PRFS_PREGS` Primary registers, including the process's P register, A and S registers, and the VL and VM registers. The structure of this address space is in the `proc_pregs` structure as defined in the `sys/fs/prfcntl.h` include file.

PRFS_VREGS Process V registers. The eight vector registers of the process appear in sequential order in this address space. Thus, the first 64 words (512 bytes) in this address space correspond to V0, and the second group of 64 words corresponds to V1.

PRFS_BREGS Process B registers. The 64 B registers of the process appear in sequential order in this address space. Thus, the word at byte offset 0 in this address space corresponds to B0; the word at byte offset 8 corresponds to B1.

PRFS_TREGS Process T registers. The 64 T registers of the process appear in sequential order in this address space. Thus, the word at byte offset 0 in this address space corresponds to T0; the word at byte offset 8 corresponds to T1.

PRFS_SM Process shared semaphores. This address space is exactly 1 word in length with the low-order bits of the word defining the state of the shared semaphores. The number of shared semaphores varies by machine architecture: 32 semaphores for CRAY Y-MP systems. This address space is read-only.

PRFS_SHRDBREGS
Process shared SB registers. The shared SB registers of the process appear in sequential order in this address space. Thus, the word at byte offset 0 in this address space corresponds to SB0; the word at byte offset 8 corresponds to SB1.

PRFS_SHRDTREGS
Process shared ST registers. The shared ST registers of the process appear in sequential order in this address space. Thus, the word at byte offset 0 in this address space corresponds to ST0; the word at byte offset 8 corresponds to ST1.

The following process address spaces are included for convenience (because they reference internal UNICOS data structures, compatibility across releases is not supported):

PRFS_PCOMM Process common structure as defined in the `sys/proc.h` include file. This address space is read-only.

PRFS_PROC Process structure as defined in the `sys/proc.h` include file. This address space is read-only.

PRFS_SESS Session table structure as defined in the `sys/session.h` include file. This address space is read-only. If the process is not in a session, any attempted read from this address space will return 0 bytes.

PRFS_UCOMM User common structure as defined in the `sys/user.h` include file. This address space is read-only.

PRFS_USER User structure as defined in the `sys/user.h` include file. This address space is read-only.

The method used to access a particular location in any of the address spaces is always the same. If the current position of the file on which a process is open is not already at the proper location, an `lseek`(2) system call should be made that identifies both the address space and the beginning byte offset within the given address space, followed by a `read`(2) or `write`(2) system call to access the data. (Alternatively, you can use the `listio`(2) system to perform both the seek and read or write operation in one system call.)

For example, the following code fragment reads the first element of the second vector register (V1) of the process open on the `fd` file descriptor:

```
#include <sys/fs/prfcntl.h>
                 :
         long   buf;
                 :
         lseek (fd, PRFS_VREGS | 64*sizeof(long), 0);
         read (fd, (char *)&buf, sizeof(long) );
```

Splitting the process address space into multiple discontinuous segments results in some slightly peculiar behavior because one I/O operation on a `/proc` file is not permitted to cross a segment boundary. Thus, I/O operations that run beyond the end of a segment are truncated.

The `/proc` files differ from other UNICOS files because various portions of the address space are always read-only (for example, `PRFS_PROC`); other UNICOS files are either entirely writable or entirely write-protected.

/proc ioctl Requests

The format for `ioctl`(2) requests to `/proc` is as follows:

```
#include <sys/fs/prfcntl.h>
ioctl (fildes, request, arg)
long *arg;
```

The valid `ioctl`(2) requests are as follows:

`PFCCSIG` If the *arg* argument is set to 0, clears all pending signals; otherwise, *arg* points to a signal mask that contains the signal numbers to be cleared.

`PFCCSIGM` If the *arg* argument is set to 0, clears all pending signals for the multitasking group; otherwise, *arg* points to a signal mask that contains the signal numbers to be cleared.

`PFCEXCLU` Marks the process text space for exclusive use. The *arg* argument is not used and should be set to 0.

`PFCGMASK` Gets the signal trace bit mask of the process. The *arg* argument must point to a long integer in which the signal trace bit mask will be returned (see `PFCSMASK`).

`PFCGMASKM` Gets the signal trace bit mask of the multitask group. The *arg* argument must point to a long integer in which the signal trace bit mask will be returned (see `PFCSMASKM`).

`PFCKILL` Sends a signal to the process. The *arg* argument must point to a long integer that contains the number of the signal to be sent.

`PFCKILLM` Sends a signal to all processes of the multitask group. The *arg* argument must point to a long integer that contains the number of the signal to be sent.

`PFCOPENT` Opens text file for reading. The *arg* argument must point to an integer in which the opened file descriptor referring to the process's text file will be returned.

`PFCREXEC` Clears the stop-on-exec flag of the process. The *arg* argument is not used and should be set to 0.

`PFCRUN` Makes the process runnable. The *arg* argument is not used and should be set to 0.

`PFCRUNM` Makes all processes of the multitask group runnable. The *arg* argument is not used and should be set to 0.

`PFCSEXEC` Sets the stop-on-exec flag of the process. The *arg* argument is not used and should be set to 0.

`PFCSMASK` Sets the signal trace bit mask of the process. The *arg* argument must point to a long integer that defines the signal trace bit mask. The process stops when any signal is received whose corresponding bit in the trace mask also is set. The trace bit mask for signal *s* is as follows:

    1L << (s-1)

`PFCSMASKM` Sets the signal trace bit mask of the multitask group. The *arg* argument must point to a long integer that defines the signal trace bit mask. When any process in the multitask group receives the signal whose corresponding bit in the trace mask also is set, the receiving process stops, and all other processes in the multitask group, are sent the `STOP` signal. The trace bit mask for signal *s* is as follows:

    1L << (s-1)

`PFCSTOP` Sends the `STOP` signal to the process. The *arg* argument is not used and should be set to 0.

`PFCSTOPM` Sends the `STOP` signal to all processes in the multitask group of which the process is a member. The *arg* argument is not used and should be set to 0.

`PFCWSTOP` Waits for the process to become stopped. You can use the *arg* argument as a pointer to an integer to which the status of the stopped process will be returned. Any status value returned in this way is interpreted in a manner identical to the status returned by the `wait`(2) system call. Alternatively, the *arg* argument can be 0, indicating that no status information will be returned.

`PFCWSTOPM` Waits for all processes in the multitask group to stop. The *arg* argument is not used and should be set to 0.

`PFCQUERYM` Returns the status of each member in the multitask group. The *arg* argument should be a pointer to a structure of type `struct pfcquery`, which contains a pointer to an array of type `struct pfcstatus` and the size of the array. The status of each task in the multitask group, up to the given maximum, is returned in the array. Each array element contains the process identifier of the task, the status of the task, and some flags. The only flag currently implemented is `PSTAT_UNKNOWN`, which means that the process has neither stopped nor exited.

`PFCSWITCHM` When debugging a multitask group, there is at any given time, a currently traced task, which is identified by its process identifier. If the *arg* argument is nonzero, it points to the process identifier, a task that is then set to be the currently traced task. Alternatively, the *arg* argument may be 0, which leaves the currently traced task unchanged. The previous currently traced tasks's PID is returned as the function value.

NOTES

The use and implementation of the `ioctl`(2) operations documented in this entry are subject to change in future releases of UNICOS.

FILES

`/proc`	`/proc` file system root
`/usr/include/sys/fs/prfcntl.h`	Definitions for the process address space

SEE ALSO

`fstab`(5)

`close`(2), `ioctl`(2), `listio`(2), `read`(2), `reada`(2), `write`(2), `writea`(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

NAME

pty – Pseudo terminal interface

IMPLEMENTATION

CRAY Y-MP systems

CRAY J90 series

CRAY EL series

DESCRIPTION

The pseudo terminal interface, pty, provides support for a device pair called a *pseudo terminal*, which is a pair of character devices. This pair consists of a master device and a slave device. The slave device provides an interface for processes that is identical to that described in termio(4). However, whereas all other devices that provide the interface described in termio(4) have a hardware device of some sort behind them, the slave device has, instead, another process manipulating it through the master half of the pseudo terminal. That is, 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 requests that apply to pseudo terminals are defined in the sys/pty.h include file, as follows:

FIONBIO Enables or disables nonblocking I/O. Nonblocking I/O is enabled by the specification (by reference) of a nonzero parameter and is disabled by a 0 parameter. When nonblocking I/O is enabled, a read or write operation returns the error EWOULDBLOCK, rather than going to sleep to wait for the input buffer to fill or the output buffer to empty.

TCIOEXT Enables or disables external processing mode. External processing allows programs that use pseudo terminals more control over echoing of data.

TCRDFL Enables "daemon read failure" mode. This mode allows the daemon to detect a read request on the master pty device without using the ioctl request TCTTRD. The daemon's read operation fails with the ENOMSG error. This error occurs whenever a read request is on both the pty and tty sides, and no data is going in either direction. When a daemon read fails with ENOMSG, the daemon should write before it issues another read request.

TCSIG Sends a signal to the client's process group; the signal sent is specified by the *arg* argument in the ioctl(2) request.

TCTTRD Returns a nonzero value if a process on the master pty device currently has an outstanding read(2) system call. The address of the word that stores the return value is specified by the *arg* argument in the ioctl(2) request.

`TIOCPKT` Enables or disables packet mode. Packet mode is enabled by the specification (by reference) of a nonzero parameter and disabled by a 0 parameter. When this request is applied to the master side of a pseudo terminal, each subsequent read operation from the terminal returns data written on the slave part of the pseudo terminal, preceded by a 0 byte (symbolically defined as `TIOCPKT_DATA`) or a single byte that reflects control status information. In the latter case, the byte is an inclusive OR of 0 or more bits. The symbolic definition of the bytes is as follows:

`TIOCPKT_FLUSHREAD` Sets whenever the read queue for the terminal is flushed.

`TIOCPKT_FLUSHWRITE` Sets whenever the write queue for the terminal is flushed.

`TIOCPKT_STOP` Sets whenever output to the terminal is stopped with <`CONTROL-s`>.

`TIOCPKT_START` Sets whenever output to the terminal is restarted.

`TIOCPKT_DOSTOP` Sets whenever `IXON` terminal control mode is enabled (see `termio`(4)).

`TIOCPKT_NOSTOP` Sets whenever `IXON` terminal control mode is disabled (see `termio`(4)).

The `rlogin`(1B) and `rlogind`(8) commands use packet mode to implement a remote login with remote echoing, local flow control with <`CONTROL-s`> and <`CONTROL-q`>, and proper back-flushing of output. Other similar programs also can use this mode.

BUGS

You cannot send an EOT to a pseudo terminal.

FILES

`/dev/pty/`*nnn*

`/dev/ttyp`*nnn*

`/usr/include/sys/pty.h`

SEE ALSO

`termio`(4), `tty`(4)

NAME

`qdd` – Physical disk device interface

IMPLEMENTATION

CRAY J90se systems

CRAY T90 systems

DESCRIPTION

The files in `/dev/qdd` are special files that allow read and write operations to physical disk devices connected to the IPN-1. Each file represents one slice of a physical disk device. The files in `/dev/qdd` are character special files that may be used directly to read and write physical disk slices. Usually, they are called to perform I/O on behalf of higher-level logical disk device drivers. For I/O on a character disk device, read and write operations must transfer multiples of the physical device sector size and all seek operations must be on physical sector size boundaries.

The files in `/dev/qdd` are not usually mountable as file systems, although you may combine one or more physical disk slices to make a mountable logical disk device (see `dsk`(4), `ldd`(4), and `mount`(8)).

The files in `/dev/qdd` are created by using the `mknod` command (see `mknod`(8)). Each must have a unique minor device number, along with other parameters used to define a physical disk slice.

The `mknod`(8) command for physical disk devices is as follows:

`mknod` *name type major minor dtype iopath start length flags altpath unit*

name Descriptive file name for the device (for example, `qdd/scr0230`).

type Type of the device data being transferred. Devices in `/dev/qdd` are character devices denoted by a `c`.

major Major device number for physical disk devices. You can specify the major number as `dev_qdd`.

minor Minor device number for this slice.

dtype Physical disk device type.

iopath The *iopath* specifies the GigaRing number the device is on, the node number to which the disk is connected, and the controller and unit number of the device The controller number is in the range 0 through 4. For array devices, the controller number is always 0. The unit number is in the range 0 through 7.

start Absolute starting block (sector) number of the slice.

length Number of blocks (sectors) in the slice.

flags Flags for physical disk device control. They are mainly used for diagnostic and maintenance purposes. Usually, the flags field should be 0 for slices in `/dev/qdd`.

altpath The optional alternate *iopath* that you can use as a back-up path to the physical disk device's second port.

unit The disk device unit number for device types that support multiple units on the same channel.

FILES

`/dev/qdd/*`

`/usr/src/c1/io/qdd.c`

SEE ALSO

`dsk`(4), `ldd`(4), `mdd`(4), `sdd`(4), `ssdd`(4) `xdd`(4)

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

NAME

`ram` – Random-access memory disk interface

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

Random-access memory (RAM) is an area of memory that you may configure as one or more character or block special devices. It is treated as a disk drive from the user level. RAM is configured in `/usr/src/uts/cf/conf.`*SN*`.c` (*SN* is the mainframe serial number); the driver uses the minor device number as an index to a slice description. The RAM interface is represented by the `/dev/ram` special file. The `ramsize` constant determines the amount of memory to be dedicated for all devices in RAM.

EXAMPLES

The following example allocates a total of 200,000 words of main memory to RAM. This example shows the allocation of two devices, each made up of 100,000 words, that could be configured in `/dev` as either character or block special files. Their minor device numbers are 0 and 1; their major device numbers depend on their location in the `bdevsw` or `cdevsw` tables.

```
#define ramsize 200000

struct size ram00[2] = {
        sliceinit(100000,0,0),
        sliceinit(100000,100000,0)
};
```

FILES

`/dev/ram`

`/usr/src/uts/cf/conf.`*SN*`.c` (*SN* is the mainframe serial number)

NAME

`rdd` – RAM disk driver

IMPLEMENTATION

Cray PVP systems with an IOS model E

DESCRIPTION

The files in `/dev/rdd` are character special files that allow read and write operations to random-access memory (RAM) disk slices. Each file represents one slice of the total RAM disk area. The total memory allocated for RAM disks is specified in the UNICOS parameter file.

Usually, I/O request lengths to RAM disks must be in 512-word multiples and start on 512-word boundaries. A RAM disk slice can assume the attributes of a physical disk device. If the sector size of the specified device consists of more than 512 words, the I/O request lengths and starts must match those for the specified device.

Usually, you cannot mount the files in `/dev/rdd` as file systems. You may specify a RAM disk slice as a whole or part of a logical disk device. You also can combine a RAM disk slice with a physical disk device. See `dsk`(4), `ldesc`(5), and `pdd`(4).

The files in `/dev/rdd` are created by using the `mknod`(8) command. Each file must have a unique minor device number, a starting block, and a length (in blocks).

The `mknod`(8) command for RAM disk devices is as follows:

`mknod` *name type major minor dtype* `0` *start length*

name Descriptive file name for the device.

type Type of the device data being transferred. Devices in `/dev/rdd` are character devices denoted by a `c`.

major Major device number for RAM disk devices. The `dev_rdd` name label in the `/uts/c1/cf/devsw.c` file denotes the major device number for RAM disk devices.

minor Minor device number for this slice. Each RAM disk slice must have a unique minor device number.

dtype (Optional) Physical disk device type. If left at 0, the RAM disk slice assumes the physical attributes of a DD-49 disk drive. For a list of physical disk device types, see `pdd`(4).

`0` Placeholder for future use.

start Absolute starting block (sector) number of the slice.

length Number of blocks (sectors) in the slice.

FILES

`/dev/rdd/*`

`/usr/src/c1/io/rdd.c`

SEE ALSO

`dsk`(4), `ldd`(4), `ldesc`(5), `pdd`(4)

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

NAME

reqt – IPI-3 interface

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The /dev/ipi3/reqt device sends the IPI-3/IPI configuration to the IPI-3/IPI packet driver, and it requests configuration, table, and device limit information.

To communicate between the packet driver and the controlling process, use the pki_ctl structure, as defined in the sys/pki_ctl.h include file. The packet driver control structure is defined as follows:

```
struct pki_ctl{
       int    pki_psigno;                  /* Signal to receive        */
       word   *pki_packet;                 /* Packet from user program */
       int    pki_nbytes;                  /* Length of packet         */
       int    pki_device;                  /* Device name              */
       }
```

The IPI-3/IPI interface uses the following ioctl(2) requests:

PKI_GET_CONFIG	Returns the IPI-3/IPI configuration
PKI_GET_DEVCONF	Returns the device configuration
PKI_GET_DEVTBL	Returns an IPI-3/IPI table
PKI_GET_OPTIONS	Returns IPI-3/IPI options
PKI_PUT_CONFIG	Sends the IPI-3/IPI configuration
PKI_SET_OPTIONS	Sets the IPI-3/IPI options

FILES

/dev/ipi3/*device-name*	IPI-3/IPI interface devices
/usr/include/sys/pki_ctl.h	Structure definition of pki_ctl

SEE ALSO

ipi3(4)

ipi3_clear(8), ipi3_config(8), ipi3_option(8), ipi3_start(8), ipi3_stat(8), ipi3_stop(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

Tape Subsystem Administration, Cray Research publication SG–2307

NAME

sdd – Striped disk driver

IMPLEMENTATION

Cray PVP systems with an IOS model E

DESCRIPTION

The files in /dev/sdd are character special files that allow read and write operations to striped disk slices. A *striped disk slice* is a logical disk device composed of two or more physical disk slices. These physical slices, also known as *members*, must be of the same physical device type and length.

Usually, I/O request lengths to striped disks must be in 512-word multiples and start on 512-word boundaries. A striped disk device assumes the physical sector size of its member physical disk devices. If the sector size of the member devices consists of more than 512 words, the I/O request lengths and starts must match those for the specified device.

Device driver level striping is used when an increase in I/O bandwidth is desired. An individual I/O request is divided into component requests, one or more for each member physical device. The basic unit of striped I/O is known as the *stripe factor*. The stripe factor is fixed based on the physical device type of the underlying members.

Usually, you cannot mount the files in /dev/sdd as file systems. You can specify a striped disk slice as a whole or part of a logical disk device. The files in /dev/sdd are all of the logical indirect type. See dsk(4), ldd(4), ldesc(5), and pdd(4).

The mknod command is used to create a striped disk inode, as follows:

mknod *name type major minor* 0 0 *path*

name Name of the logical device.

type Type of the device data being transferred. Devices in /dev/sdd are character devices denoted by a c.

major Major device number of the striped logical disk device driver. The name dev_sdd in the /usr/src/uts/c1/cf/devsw.c file denotes the driver.

minor Minor device number for this slice. Each striped disk slice must have a unique minor device number.

0 0 Placeholders for future use.

path Path name that designates the logical descriptor file listing the member slices. See ldesc(5).

EXAMPLES

The following example creates a striped disk inode:

```
mknod /dev/sdd/usr c dev_sdd 1 0 0 /dev/ldd/usr.stripe
```

FILES

`/dev/sdd/*`

`/usr/include/sys/sdd.h`

`/usr/src/c1/io/sdd.c`

SEE ALSO

`dsk`(4), `ldd`(4), `ldesc`(5), `pdd`(4)

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

NAME

`sds` – Secondary data storage interface on SSD devices

IMPLEMENTATION

Cray PVP systems (except CRAY J90 series)

DESCRIPTION

The secondary data storage (SDS) device is extended storage space allocated on an SSD (solid-state storage device). It can be used by users for extended storage, or by the operating system for use as logical device cache. See `ssd`(4) and `ldcache`(8).

Users can allocate SDS space using the `ssbreak`(2) system call. Reads and writes of SDS space can use the specialized `ssread`(2) and `sswrite`(2) UNICOS system calls, or the more general purpose `read`(2), `write`(2), `reada`(2), `writea`(2), and `listio`(2) systems calls.

The `ssread`(2) and `sswrite`(2) system calls do not require a file descriptor. There is only one SDS device and only an `ssbreak`(2) system call is required to allocate extended storage before the `ssread`(2) or `sswrite`(2) system calls. The `ssread`(2) and `sswrite`(2) system calls, however, are limited to synchronous operation. See `ssread`(2) and `sswrite`(2).

The character special file, `/dev/sds`, provides a general purpose interface to the SDS device. By opening `/dev/sds`, a file descriptor is obtained to allow `read`(2), `reada`(2), `write`(2), `writea`(2), and `listio`(2) system calls. File permissions on `/dev/sds` allow any user to open it; however, an `ssbreak` is required to allocate space before any reads or writes are allowed.

The character special file, `/dev/sds` is created by the `mknod`(8) command as follows:

```
mknod /dev/sds c dev_sds 0
```

FILES

`/dev/sds`

SEE ALSO

`ssd`(4), `ssdd`(4), `ssdt`(4)

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

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

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

NAME

secded – SECDED maintenance function interface

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

The SECDED maintenance functions allow access to the memory error correction interface of the hardware. These functions allow the setting and clearing of data bits or check bits in a word of memory or a processor register. Reading a word that has been set in this way reveals whether memory error detection, correction, and reporting are working properly. The functions also provide a means of controlling the scrubbing of single-bit memory errors. The parameters that control the system's response to bursts of memory errors can be manipulated through this interface.

The secded driver supports the ioctl(2) and open(2) system calls. The driver supports only one device (minor device 0).

The following ioctl requests are accepted:

ME_GET
: Accepts as an argument a pointer to a structure that the driver will fill with the memory error correction parameters *mecormax*, maximum number of single-bit errors that can occur in *meint* period of time with no intervals of longer than *meint/mecormax* seconds without an error, then single-bit error detection is turned off for all user processes for *medisint* seconds. The last parameter, *meuncmax*, is the limit of uncorrectable errors that the UNICOS system will allow before forcing a panic because of the memory errors. The structure is defined in sys/memc.h. The default parameters are defined by MECORMAX as 16 errors, MEINT as 5*HZ or 5 seconds, MEDISINT as 300 or 60 seconds times 5, and MEUNCMAX as 64.

ME_SET
: Accepts as an argument a pointer to a structure that the driver will extract the new memory error correction parameters, *meint*, *mecormax*, *medisint*, and *meuncmax*.

RPE_SET
: Accepts as an argument a pointer to a structure that contains a register set designator (RPE_V, RPE_T, RPE_B, RPE_IB, or RPE_SR) and a parity indicator (even or odd).

: RPE_SET allows the CPU register parity error functions to be tested. Incorrect even or odd register parity may be written into a V, T, or B register; a shared register; or an instruction buffer, and then read to force a register parity error interrupt.

: Special maintenance instructions used for these functions exist only on CRAY Y-MP CPUs that have a revision level of 4 or later. These instructions behave as NO-OPs on other CPUs. Currently, only the V register function (RPE_V) is supported. The RPE_T, RPE_B, RPE_IB, and RPE_SR functions are deferred.

`SD_SET` Accepts as an argument a pointer to a structure that contains the address of the word to be modified and the data and check bits to be modified within the word.

The `ioctl` request uses the SECDED maintenance instructions to read the entire word (64 data and 8 check bits), complements the data and check bits to be modified, and rewrites the entire 72-bit data word to memory. In this way, any 72-bit pattern, including the associated check bits, can be placed into a memory word.

`CPU_GET` Accepts as an argument a pointer to a structure that the driver will fill with the parameters controlling downing of CPUs on uncorrectable memory errors: *umemax*, *umelife* and *umedown*. *umemax* is the number of uncorrectable memory errors per CPU that can occur before the CPU is downed by the operating system. Each error has a lifetime of *umelife* seconds, after which it is no longer counted in the number of errors for a CPU. The CPU will remain down for *umedown* seconds, after which it will automatically be returned to service by the operating system.

Setting *umemax* to zero disables the automatic downing of the CPU. Setting *umedown* to zero disables the automatic return to service of the CPU. The defaults are zero for *umemax* and *umedown* and 86400 (24 hours) for *umelife*.

`CPU_SET` Accepts as an argument a pointer to a structure from which the driver will extract the new parameters to control the downing of CPUs on uncorrectable memory errors: *umemax*, *umelife* and *umedown*.

NOTES

For the maintenance functions to work, the error maintenance switch on the mainframe switch panel must be enabled. The software, however, cannot determine whether the switch is enabled. For memory error detection and correction to work properly, the error correction switch on the mainframe also must be on (this is the normal state). Because of the nature of the SECDED maintenance instructions, any test using this capability should be done in single-user mode. The address of a word being modified is checked only to ensure that it is within the physical memory of the machine. If a word is changed to contain a double-bit or multibit error, and the kernel reads that word next (as opposed to a user process reading that word next), the kernel panics.

For CRAY Y-MP systems, you must deadstart the mainframe with the maintenance mode switch in the off position. After the mainframe is deadstarted (and the system is running in single-user mode), you should turn the maintenance mode switch to the on position.

MESSAGES

The following errors can occur:

`EFAULT` Returned if the word to be modified is outside the machine's memory, or if the parameter structure is outside the user's field length.

`ENXIO` Returned if other than minor device 0 is selected, or if the mainframe is not an appropriate type for the attempted operation.

EPERM Returned if the user is not the super user.

EINVAL Returned if parameters passed in on SD_SET, ME_SET, RPE_SET or CPU_SET do not pass validation tests.

FILES

/dev/secded

/usr/include/sys/memc.h

/usr/include/sys/secded.h

SEE ALSO

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

NAME

`sfs` – File that contains the names of each Cray Research system in an SFS cluster and its associated SFS arbiter

SYNOPSIS

`/etc/config/sfs`

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

The configuration of SFS arbiters is represented in the `/etc/config/sfs` file. The `/etc/config/sfs` file is generated using the menu system, or created directly with a text editor. The `/etc/config/sfs` file should be identical on all systems within a cluster that share file systems.

The format of the `/etc/config/sfs` file is line oriented. Each line begins with one of two valid keywords starting in column 1: `HostName` or `Arbiter`.

The `HostName` Line

The `HostName` line describes the name of a system, and denotes which SFS arbiters are valid and accessible by that system. For example, the following line defines that the host `frost` is a system within this cluster, and that `frost` can access SFS arbiters 0, 1, and 2, as further defined within `/etc/config/sfs`:

```
HostName        frost   0,1,2
```

The `Arbiter` Line

The `Arbiter` line describes the identity of an SFS arbitration service, and the path names of the three character special devices necessary to support that arbitration service. For example, the following line defines an SFS arbiter with a numeric identity of 0, with a symbolic name of SMP-2, and which uses the three path names `/dev/smp-0`, `/dev/sfs-0`, and `/dev/smnt-0` to access the three character special devices that define an SFS arbiter:

```
Arbiter              0  SMP-2        /dev/smp-0 /dev/sfs-0 /dev/smnt-0
```

The first path name describes the character special for the physical semaphore device. For example, the output of file `/dev/smp-0` may look like the following, which describes an SMP-2 (device type equals 2) attached to low-speed channel pair 18:

```
/dev/smp-0:     character special (73/0)    2 18 0 0 0 0 0 0
```

The output of file `/dev/smp-0` also may look like the following, which describes an H-SMP (device type equals 4) representing port 7, whose HIPPI I/O path is described in the character special node `/dev/hdd/smp`:

```
/dev/smp:        character special (73/0)    4 7 /dev/hdd/smp 0 0
```

The second path name describes the character special for the logical shared file system driver, and its associated Shared Lock Region. For example, the output of file `/dev/sfs-0` may look like the following, which describes an interface to the logical SFS driver that uses `dev/dsk/slr` as the shared medium necessary to communicate semaphore allocation and other shared information to the other systems in the cluster:

```
/dev/sfs-0:      character special (48/0)    0 0 /dev/dsk/slr 0 0
```

The third path name describes the character special for the Shared Mount Table interface. For example, the output of file `/dev/smnt-0` may look like the following, which describes an interface to the logical SFS driver that uses a portion of the shared medium described in `/dev/sfs-0` as a record of SFS mounts to be shared with the other systems in the cluster:

```
/dev/smnt-0:     character special (75/0)    0 0 0 0 0 0 0 0
```

The minor number assigned to each of the three character special devices must be the same, and it must match the SFS arbiter numeric identity defined in the `/etc/config/sfs` file.

EXAMPLES

An example of `/etc/config/sfs` taken from a test system looks like the following:

```
c
o
l
u
m
n

1
|
v
HostName        frost   0,1,2
HostName        ice     0,1,2
HostName        sn5609  1,2
Arbiter         0  SMP-2        /dev/smp-0 /dev/sfs-0 /dev/smnt-0
Arbiter         1  HSMP         /dev/smp-1 /dev/sfs-1 /dev/smnt-1
Arbiter         2  Simulator    /dev/smp-2 /dev/sfs-2 /dev/smnt-2
```

FILES

`/dev/sfs`	External Semaphore Device Logical-layer Interface
`/dev/smp`	Low-level interface to the semaphore device
`/dev/smnt`	Shared mount table interface

SEE ALSO

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

Shared File System (SFS) Administrator's Guide, Cray Research publication SG–2114

NAME

`slog` – Security log interface

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `/dev/slog` pseudo device is a read-only device that holds security log records. The security log daemon, `slogdemon`(8), transfers those records to the security log file for use with the security utilities. For more information about the security log file, see `slrec`(5).

FILES

`/dev/slog`	Security log pseudo device
`/usr/adm/sl/slogfile`	Disk-resident security log

SEE ALSO

`slrec`(5)

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

General UNICOS System Administration, Cray Research publication SG–2301

NAME

ssd – Solid state storage device

IMPLEMENTATION

Cray PVP systems (except CRAY J90 series)

DESCRIPTION

The SSD solid-state storage device is a high-speed secondary memory available on Cray PVP systems (except CRAY J90 series).

You can configure the SSD as a disk device used for filesystems or as a secondary data storage (SDS) device. See ssdd(4), ssdt(4), and sds(4). SDS space can be used for extended storage or can be configured as logical device cache with the ldcache(8) command. For more information, see ldcache(8).

When configured as a disk, the SSD functions as a fast random-access device that can be used for mounting file systems or for swapping. In this case, the SSD is represented in /dev/ssdd by one or more files. For more information about this configuration of the SSD, see ssdd(4) for IOS model E based systems or ssdt(4) for GigaRing-based systems.

FILES

/dev/dsk

/dev/sds

/dev/ssdd

/dev/ssdt

SEE ALSO

sds(4), ssdd(4), ssdt(4)

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

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

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

NAME

`ssdd` – SSD disk driver

IMPLEMENTATION

Cray PVP systems with an IOS model E

DESCRIPTION

The files in `/dev/ssdd` are character special files that allow read and write operations to SSD solid-state disk slices. Each file represents a one slice of the total SSD disk area. The total amount of the SSD allocated for SSD disks is specified in the UNICOS parameter file.

Usually, I/O request lengths to SSD disks must be in 512-word multiples and start on 512-word boundaries. A SSD disk slice can assume the attributes of a physical disk device. If the sector size of the specified device consists of more than 512 words, the I/O request lengths and starts must match those for the specified device.

Usually, you cannot mount the files in `/dev/ssdd` as file systems. You may specify a SSD disk slice as a whole or part of a logical disk device. You also can combine a SSD disk slice with a physical disk device. See `dsk`(4), `ldesc`(5), and `pdd`(4).

The files in `/dev/ssdd` are created by using the `mknod` command (see `mknod`(8)). Each must have a unique minor device number, a starting block, and a length (in blocks).

The `mknod`(8) command for SSD disk devices is as follows:

`mknod` *name type major minor dtype* `0` *start length*

name Descriptive file name for the device.

type Type of the device data being transferred. Devices in `/dev/ssdd` are character devices denoted by a `c`.

major Major device number for SSD disk devices. The `dev_ssdd` name label in the `/uts/c1/cf/devsw.c` file denotes the major device number for SSD disk devices.

minor Minor device number for this slice. Each SSD disk slice must have a unique minor device number.

dtype Physical disk device type. This is optional. If left at 0, the SSD disk slice assumes the physical attributes of a DD-49 disk drive. For a list of physical disk device types, see `pdd`(4).

`0` Placeholder for future use.

start Absolute starting block (sector) number of the slice.

length Number of blocks (sectors) in the slice.

FILES

`/dev/ssdd/*`

`/usr/src/c1/io/ssdd.c`

SEE ALSO

`dsk`(4), `ldd`(4), `ldesc`(5), `pdd`(4)

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

NAME

ssdt – GigaRing-based Solid State Disk storage device interface

IMPLEMENTATION

CRAY T90 systems

DESCRIPTION

The files in /dev/ssdt are special files that allow read and write operations to the GigaRing-based Solid State Disk storage device known as the SSD-T90. Each file represents one slice of an SSD-T90. The files in /dev/ssdt are character special files that may be used directly to read and write physical SSD-T90 slices.

You can configure the SSD-T90 as a disk device or as a secondary data storage (SDS) device.

When configured as a disk, the SSD-T90 functions as a fast random-access device that can be used for mounting file systems or for swapping. In this case, the SSD-T90 is represented in /dev/ssdt by one or more files. Usually, they are called to perform I/O on behalf of higher-level logical disk device drivers.

An iounit is a multiple of 4096-byte blocks that corresponds to the smallest read or write possible to a character special disk device. The iounit for an SSD-T90 device is normally 1, meaning read and write operations must transfer multiples of 4096 bytes and all seek operations must be on 4096-byte boundaries. The iounit may be set at a value greater than 1 as described below in the mknod command detail.

The files in /dev/ssdt are not usually mountable as file systems, although you may combine one or more physical disk slices to make a mountable logical disk device (see dsk(4), ldd(4), and mount(8)).

When configured as Secondary Data Storage (SDS), the SSD-T90 is managed in much the same way as main memory. It can be accessed directly by users with the ssbreak(2), ssread(2), and sswrite(2) system calls, or allocated as logical device cache for the caching of filesystem data. See ssbreak(2), ssread(2), sswrite(2), and ldcache(8).

The files in /dev/ssdt are created by using the mknod(8) command (see mknod(8)). Each must have a unique minor device number, along with other parameters used to define a physical disk slice.

The mknod(8) command for physical disk devices is as follows:

mknod *name type major minor dtype iopath start length flags reserved unit*

name Descriptive file name for the device (for example, /dev/ssdt/ssdt_blk0).

type Type of the device data being transferred. Devices in /dev/ssdt are character devices denoted by a c.

major Major device number for physical disk devices. The dev_ssdt name label in the /usr/src/uts/c1/cf/devsw.c file denotes the major device number for physical disk devices. You can specify the major number as dev_ssdt.

minor Minor device number for this slice.

dtype The *dtype* field is a compound field containing the `iounit` and target memory type for the SSD-T90. Target memory types are defined in the include file `sys/tmio.h`. The *dtype* field is broken down in octal as follows:

0*tttiiii* where:

ttt = the target memory type

iiii = the `iounit`

The SSD-T90 is either an 8 or 16 processor CRAY T3E. The value of the *dtype* field should be 0100001 for an 8 processor or 0110001 for a 16 processor T3E.

iopath The *iopath* specifies the GigaRing ring and node number that the SSD-T90 is connected to. It contains the following fields when broken down in octal:

0*rrrnn*0 where:

rrr = GigaRing ring number

nn = GigaRing node number

start Absolute starting block (`iounit` multiple) number of the slice.

length Number of blocks (`iounit` multiple) in the slice.

flags Flags for physical disk device control. They are mainly used for diagnostic and maintenance purposes. Usually the *flags* field should be 0 for slices in */dev/ssdt*.

reserved Currently unused. Should be 0.

unit Designates the SSD-T90 unit number. If only one SSD existis on a system, the unit should be 0.

Further information about configuring SSD-T90 for use as SDS memory can be found in *UNICOS Configuration Administrator's Guide*, publication SG-2303.

FILES

`/dev/ssdt/*`

`/usr/include/sys/ssdt.h`

`/usr/src/c1/io/ssdt.c`

SEE ALSO

`dsk`(4), `ldd`(4), `mdd`(4), `qdd`(4), `sdd`(4), `ssdd`(4),

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

NAME

`tape` – Physical tape device interface

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

Each tape device node in the `/dev/tape` directory provides an interface to a real physical tape device.

These devices attach to Cray Research systems with I/O subsystems model E using IBM Block Mux channels, ANSI intelligent peripheral interface (IPI) channels, Enterprise Systems Connection (ESCON) channels, or Small Computer System Interface (SCSI) channels. They attach to Cray Research GigaRing based systems using IBM Block Mux channels or SCSI channels.

The physical tape driver interface translates requests from a user or the tape daemon into requests packets that are sent to the attached I/O subsystem. The type determines the type of physical devices that may be attached. It manipulates the physical interface to accomplish the requested function and returns status to the system.

During system start-up, a file describing the tape configuration (`/etc/config/text_tapeconfig`) is read, tape device nodes are created in the `/dev/tape` directory, the configuration is sent to the tape driver and related I/O processors (IOPs), and the channels and control units are configured to the state specified in the tape configuration file. The permissions on these device paths are generally reserved for the root account and will have appropriate security labels.

You have two interfaces available for accessing tape devices:

Interface	**Description**
Character-special tape	Provides unstructured access to tape devices. Its capabilities provide tape access similar to the access that users on other UNIX systems have. This access is a basis means of reading and writing tape information.
Tape daemon-assisted (`tpddem`(4))	Intercepts user system call requests and processes requests from tape-related commands to perform tape resource management, device management, volume mounts and dismounts through operator communications or autoloader requests, label processing, volume switching, and error recovery. This interface is called the Tape Management Facility. The character-special tape interface does not provide these capabilities.

If the tape daemon-assisted interface is needed, executing the `tpdaemon`(8) command creates the daemon process that provides this interface. The tape daemon-assisted interface can use the configuration established during system start-up, or it can redefine the configuration. It operates concurrently with the character-special tape interface.

FILES

`/dev/tape/`*device_name*	Tape device node
`/etc/config/text_tapeconfig`	Tape subsystem configuration file

SEE ALSO

`tpddem`(4), `text_tapeconfig`(5)

Tape Subsystem User's Guide, Cray Research publication SG–2051

Tape Subsystem Administration, Cray Research publication SG–2307

NAME

termio, termios – General terminal interface

SYNOPSIS

```
#include <termio.h>
ioctl (int fildes, int request, struct termio *arg);
ioctl (int fildes, int request, int arg);

#include <termios.h>
ioctl (int fildes, int request, struct termios *arg);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

All of the asynchronous communications ports use the same general interface, no matter what hardware is involved. The remainder of this entry discusses the common features of this interface.

A terminal is associated with a terminal file in `/dev`. Terminal file names have the following form:

```
/dev/tty*
```

The user interface to this functionality is through function calls (the preferred interface) described on `terminal`(3C) man page or by the `ioctl`(2) requests described in this entry. This entry also discusses the common features of the terminal subsystem that are relevant to both user interfaces.

When a terminal file is opened, it usually causes the process to wait until a connection is established. In practice, a user's programs seldom open terminal files; they are opened by `getty`(8) and become a user's standard input, standard output, and standard error files. The very first terminal file opened by the session leader that is not already associated with a session becomes the controlling terminal for that session. The controlling terminal plays a special role in handling quit and interrupt signals; that role is discussed in this entry. The controlling terminal is inherited by a child process during a `fork`(2) system call. A process can break this association by changing its session using the `setsid`(2) system call.

A terminal associated with one of these files ordinarily operates in full-duplex mode. Characters may be typed at any time, even while output is occurring, and are lost only when the character input buffers of the system become completely full (for example, if the user has accumulated `MAX_INPUT` number of input characters that have not yet been read by some program); this situation is rare. (When the input limit is reached, all of the characters saved in the buffer up to that point are deleted without notice.)

Session Management (Job Control)

When a session is associated with a terminal, the control terminal designates one of the process groups as the foreground process group; all other process groups in the session are designated as background process groups. The foreground process group plays a special role in handling signal-generating input characters, as discussed in this entry. By default, when a controlling terminal is allocated, the controlling process's process group is assigned as foreground process group.

Background process groups in the controlling process's session are subject to a job control line discipline when they try to access their controlling terminal. Process groups can be sent signals that will cause them to stop, unless they have made other arrangements. An exception is made for members of orphaned process groups. These are process groups that do not have a member with a parent in another process group that is in the same session and therefore shares the same controlling terminal. When a member's orphaned process group tries to access its controlling terminal, errors will be returned because there is no process to continue it if it should stop.

If a member of a background process group tries to read its controlling terminal, its process group sent a `SIGTTIN` signal, which usually causes the members of that process group to stop. If, however, the process is ignoring or holding `SIGTTIN`, or is a member of an orphaned process group, the read operation will fail with an `errno` value set to `EIO`, and no signal will be sent.

If a member of a background process group tries to write its controlling terminal and the `TOSTOP` bit is set in the `c_lflag` field, its process group will be sent a `SIGTTOU` signal, which usually causes the members of that process group to stop. If, however, the process is ignoring or holding `SIGTTOU`, the write operation will succeed. If the process is not ignoring or holding `SIGTTOU` and is a member of an orphaned process group, the write operation will fail with `errno` set to `EIO`, and no signal will be sent.

If `TOSTOP` is set and a member of a background process group tries to issue an `ioctl`(2) system call to its controlling terminal, and that `ioctl` will modify terminal parameters (for example, `TCSETA`, `TCSETAW`, `TCSETAF`, or `TIOCSPGRP`), its process group will be sent a `SIGTTOU` signal, which usually causes the members of that process group to stop. If, however, the process is ignoring or holding `SIGTTOU`, the `ioctl`(2) will succeed. If the process is not ignoring or holding `SIGTTOU` and is a member of an orphaned process group, the write will fail with `errno` set to `EIO`, and no signal will be sent.

Canonical Mode Input Processing

Typically, all line editing and echoing functions are performed by the Cray Research system. You can off load this processing to the front-end system when using `telnet`(1B); however, not all of the features described in this entry are available in this mode.

Usually, terminal input is processed in units of lines. A line is delimited by a newline character (ASCII LF), an end-of-file character (ASCII EOT), or an end-of-line character. This means that a program trying a read operation is suspended until an entire line has been typed. Also, no matter how many characters are requested in the read operation, at most one line is returned. However, a whole line does not have to read at once; any number of characters may be requested in a read operation without loss of information.

Erase and kill processing is usually done during input. The `ERASE` character (by default, `#`) erases the last character typed. The `WERASE` character (`CONTROL-w`) erases the last *word* typed in the current input line (but not any preceding spaces or tabs). A *word* is defined as a sequence of nonblank characters, with tabs counted as blanks. Neither `ERASE` nor `WERASE` erases beyond the beginning of the line. The `KILL` character (by default, `@`) kills (deletes) the entire input line, and optionally outputs a newline character. All of these characters operate on a keystroke basis, independently of any backspaces or tabs that may have been entered. The `REPRINT` character (`CONTROL-r`) prints a newline character, followed by all characters that have not been read. Reprinting also occurs automatically if characters that usually would be erased from the screen are garbled by program output. The characters are reprinted as if they were being echoed; consequencely, if `ECHO` is not set, they are not printed.

You may enter both the erase and kill characters literally if they are preceded by the escape `\` symbol. In this case, the escape character is not read. You may change the erase and kill characters.

Noncanonical Mode Input Processing

In noncanonical mode input processing, input characters are not assembled into lines. Erase and kill processing does not occur. The `MIN` and `TIME` values are used to determine how to process the characters received, as follows:

`MIN` Minimum number of characters that should be received when the read is satisfied (that is, when the characters are returned to the user).

`TIME` Timer of 0.10-second granularity used to time-out transmissions that occur in bursts and short-term data transmissions.

The four possible combinations for `MIN` and `TIME` and their interactions are as follows:

Case A: `MIN` > 0, `TIME` > 0

In this case, `TIME` serves as an intercharacter timer and is activated after the first character is received. Because it is an intercharacter timer, it is reset after a character is received. The interaction between `MIN` and `TIME` is as follows. As soon as one character is received, the intercharacter timer is started. If `MIN` characters are received before the intercharacter timer expires (the timer is reset on receipt of each character), the read is satisfied. If the timer expires before `MIN` characters are received, the characters received to that point are returned to the user. If `TIME` expires, at least one character will be returned, because the timer would not have been enabled unless a character was received. In this case (`MIN` > 0, `TIME` > 0), the read sleeps until the `MIN` and `TIME` mechanisms are activated by the receipt of the first character. If the number of characters read is fewer than the number of characters available, the timer is not reactivated, and the subsequent read is satisfied immediately.

Case B: `MIN` > 0, `TIME` = 0

In this case, because the value of `TIME` is 0, the timer plays no role and only `MIN` is significant. A pending read is not satisfied until `MIN` characters are received (the pending read sleeps until `MIN` characters are received). A program that uses this case to read record-based terminal I/O may be blocked indefinitely in the read operation.

Case C: MIN = 0, TIME > 0

In this case, because MIN = 0, TIME no longer represents an intercharacter timer; it now serves as a read timer that is activated as soon as one read operation is requested. A read request is satisfied as soon as one character is received or the read timer expires. In this case, if the timer expires, no character is returned. If the timer does not expire, the read can be satisfied only if a character is received. In this case, the read will not block indefinitely waiting for a character; if no character is received within TIME*.10 seconds after the read is initiated, the read returns with zero characters.

Case D: MIN = 0, TIME = 0

In this case, return is immediate. The minimum of either the number of characters requested or the number of characters currently available is returned without waiting for more characters to be input.

The remainder of this subsection compares the different cases of interaction between the MIN and TIME values. In the following explanations, the interactions of MIN and TIME are not symmetric. For example, when MIN is greater than 0 and TIME equals 0, TIME has no effect. However, in the opposite case, in which MIN equals 0 and TIME is greater than 0, both MIN and TIME play a role in that MIN is satisfied with the receipt of one character. In case A (MIN greater than 0, TIME greater than 0), TIME represents an intercharacter timer; in case C (TIME equals 0, TIME greater than 0), TIME represents a read timer.

These two points highlight the dual purpose of the MIN/TIME feature. Cases A and B, in which MIN > 0, exist to handle burst mode activity (for example, file transfer programs), where a program would like to process at least MIN characters at a time. In case A, the intercharacter timer is activated by a user as a safety measure; in case B, the timer is turned off.

Cases C and D exist to handle single character, timed transfers. These cases are readily adaptable to screen-based applications that must know whether a character is present in the input queue before refreshing the screen. In case C, the read is timed; in case D, it is not.

Another important note is that MIN is always just a minimum. It does not denote a record length (for example, if a program does a read of 20 bytes, MIN is 10, and 25 characters are present, 20 characters will be returned to the user). When one or more characters are written, they are transmitted to the terminal as soon as previously-written characters have finished typing. If echoing has been enabled, input characters are echoed as they are typed. If a process produces characters more rapidly than they can be typed, it will be suspended when its output queue exceeds some limit. When the queue is drained down to some threshold, the program is resumed. Certain characters have special functions on input. These functions and their default character values are summarized as follows:

DISCARD (CONTROL-o or ASCII SI) describes subsequent output. Output is discarded until you type another DISCARD character, more input arrives, or a program you type clears the condition.

DSUSP (CONTROL-y or ASCII EM) generates a suspend (SIGTSTP) signal, such as SUSP, but the signal is sent when a process in the foreground process group tries to read the DSUSP character, rather than when the character is typed.

EOF (CONTROL-d or ASCII EOT) generates an end-of-file from a terminal. When this character is received, all of the characters that are waiting to be read are passed immediately to the program, without waiting for a newline character, and the EOF is discarded. If no characters are waiting (that is, the EOF occurred at the beginning of a line), 0 characters, the standard end-of-file indication, is passed back. The EOF character is not echoed unless it is escaped. Because ASCII EOT is the default EOF character, this prevents terminals that respond to EOT from hanging up.

EOL (ASCII NUL) is an additional line delimiter, such as NL. Usually, it is not used.

EOL2 An additional line delimiter, such as NL. Usually, it is not used.

ERASE (RUBOUT or ASCII DEL) erases the preceding character. It does not erase beyond the start of a line, as delimited by a NL, EOF, EOL, or EOL2 character.

INTR (CONTROL-c or ASCII ETX) generates an interrupt (SIGINT) signal that is sent to all frequent processes associated with the controlling terminal. Usually, each such process is forced to terminate, but arrangements may be made either to ignore the signal or to receive a trap to an agreed-upon location; see signal(2).

KILL (CONTROL-u or ASCII NAK) deletes the entire line, as delimited by a NL, EOF, EOL, or EOL2 character.

LNEXT (CONTROL-v or ASCII SYN) ignores the special meaning of the next character. This works for all of the special characters mentioned in this list. It allows characters to be input that would otherwise be interpreted by the system (for example, KILL or QUIT).

NL (ASCII LF) is the normal line delimiter, but you can escape it by using the LNEXT character.

QUIT (CONTROL-| or ASCII FS) generates a quit (SIGQUIT) signal. Its treatment is identical to the interrupt (SIGINT) signal, except that unless a receiving process has made other arrangements, it is terminated, and a core image file (called core) is created in the current working directory.

REPRINT (CONTROL-r or ASCII DC2) reprints all characters, preceded by a newline character, that have not been read.

START (CONTROL-q or ASCII DC1) resumes output that has been suspended by a STOP character. While output is not suspended, START characters are ignored.

STOP (CONTROL-s or ASCII DC3) suspends output temporarily. It is useful with terminals to prevent output from disappearing before it can be read. While output is suspended, STOP characters are ignored.

SUSP (CONTROL-z or ASCII SUB) generates a suspend (SIGTSTP) signal, which stops all processes in the foreground process group for that terminal.

SWTCH (ASCII NUL) is reserved for future use.

WERASE (CONTROL-w or ASCII ETX) erases the preceding word. (A *word* is defined as a sequence of nonblank characters, with tabs counted as blanks.) It does not erase beyond the start of a line, as delimited by a NL, EOF, EOL, or EOL2 character.

You may change all character values except NL to suit individual tastes. If the value of a special control character is _POSIX_VDISABLE (0), the function of that special control character is disabled. To escape the ERASE, KILL, and EOF characters, use a preceding \ symbol; in which case, no special function is performed. You can precede any of the special characters by the LNEXT character; in which case, no special function is performed.

Modem Disconnect

When a modem disconnect is detected, a hang-up (SIGHUP) signal is sent to the terminal's controlling process. Unless other arrangements have been made, these signals terminate the process. If SIGHUP is ignored or caught, any subsequent read operation returns with an end-of-file (EOF) indication until the terminal is closed.

If the controlling process is not in the foreground process group of the terminal, a SIGTSTP is sent to the terminal's foreground process group. Unless other arrangements have been made, these signals stop the processes.

Processes in background process groups that try to access the controlling terminal after modem disconnect while the terminal is still allocated to the session will receive appropriate SIGTTOU and SIGTTIN signals. Unless other arrangements have been made, this signal stops the processes.

The controlling terminal remains in this state until it is reinitialized with a successful open by the controlling process, or deallocated by the controlling process. The parameters that control the behavior of devices and modules providing the termios interface are specified by the termios structure defined by the sys/termios.h include file. Several ioctl(2) requests apply to terminal files. The primary requests use the following structure, defined in the sys/termios.h include file:

```
tcflag_t     c_iflag;          /* input modes   */
tcflag_t     c_oflag;          /* output modes  */
tcflag_t     c_cflag;          /* control modes */
tcflag_t     c_lflag;          /* local modes   */
cc_t         c_cc[NCCS];       /* control characters */
```

The c_cc array defines the special control characters. The symbolic name NCCS is the size of the control-character array and also is defined by termios.h. The relative positions, subscript names, and typical default values for each function are as follows:

0	VINTR	DEL
1	VQUIT	FS
2	VERSE	#
3	VKILL	@
4	VEOF	EOT
5	VEOL	NUL
6	VEOL2	NUL
7	VSWTCH	NUL
8	VSTRT	DC1
9	VSTOP	DC3
10	VSUSP	SUB
11	VDSUSP	EM
12	VREPRINT	DC2
13	VDISCRD	SI
14	VWERSE	ETB
15	VLNEXT	SYN
16-19	Reserved	

Input Modes

The `c_iflag` field describes the basic terminal input control, as follows:

IGNBRK	Ignores break condition
BRKINT	Signals interrupt on break
IGNPAR	Ignores characters with parity errors
PARMRK	Marks parity errors
INPCK	Enables input parity check
ISTRIP	Strips character
INLCR	Maps NL to CR on input
IGNCR	Ignores CR
ICRNL	Maps CR to NL on input
IUCLC	Maps uppercase to lowercase on input
IXON	Enables start and stop output control
IXANY	Enables any character to restart output
IXOFF	Enables start and stop input control

The initial input control value is BRKINT, IGNPAR, ISTRIP IXON, and IXANY. If set, the bits have the following meanings:

`IGNBRK` Ignores the break condition (a character-framing error with data that consists of all 0's); that is, nothing is put on the input queue and therefore, a process does not read any break character. Otherwise, if `BRKINT` is set, the break condition flushes the input and output queues and, if the terminal is the controlling terminal of a foreground process group, sends the interrupt (`SIGINT`) signal to that foreground process group. Otherwise, if neither `IGNBRK` nor `BRKINT` is set, a break condition is read as a single ASCII NULL character (´\0´) (if `PARMRK` is set, it is read as as ´\377´, ´\0´, ´\0´).

`BRKINT` Generates an interrupt signal for the break condition and flushes both the input and output queues if `IGNBRK` is set.

`IGNPAR` Ignores bytes that have framing or parity errors (other than break).

`PARMRK` Reads any character that has a framing or parity error, other than break, that is not ignored (`IGNPAR` is not set) as the 3-character sequence 0377, 0, *X*; *X* is the data of the character received in error. To avoid ambiguity in this case, a valid character of 0377 is read as 0377, 0377 if `ISTRIP` is not set. If neither `IGNPAR` nor `PARMRK` is set, a character that has a framing or parity error (other than break) is read as a single ASCII NULL character (´\0´).

`INPCK` Enables input parity checking. If `INPCK` is not set, input parity checking is disabled. This allows output parity generation without input parity errors. Whether input parity checking is enabled or disabled is independent of whether parity detection is enabled or disabled. If parity detection is enabled but input parity checking is disabled, the hardware to which the terminal is connected will recognize the parity bit, but the terminal special file will not check whether this is set correctly.

`ISTRIP` Strips valid input characters to 7 bits; if `ISTRIP` is not set, all 8 bits are processed.

`INLCR` Translates a received `NL` character into a `CR` character. If `IGNCR` is set, a received `CR` character is ignored; otherwise, if `ICRNL` is set, a received `CR` character is translated into a `NL` character.

`IGNCR` Ignores a received `CR` character. If `IGNCR` is not set, and `ICRNL` is set, a received `CR` character is translated into an `NL` character.

`ICRNL` Translates a received `CR` character into an `NL` character if `IGNCR` is not set.

`IUCLC` Translates a received uppercase alphabetic character into the corresponding lowercase character.

`IXON` Enables start and stop output control; a received `STOP` character suspends output, and a received `START` character restarts output. The `STOP` and `START` characters are not read, but they merely perform flow control functions. If `IXANY` is set, any input character restarts output that has been suspended.

`IXANY` Any input character restarts suspended output.

`IXOFF` Transmits a `START` character when the input queue is nearly empty and a `STOP` character when the input queue is nearly full.

Output Modes

The `c_oflag` field specifies the system treatment of output, as follows:

`OPOST`	Postprocesses output
`OLCUC`	Maps lowercase to uppercase on output
`ONLCR`	Maps `NL` to `CR-NL` on output
`OCRNL`	Maps `CR` to `NL` on output
`ONOCR`	No `CR` output at column 0
`ONLRET`	`NL` performs `CR` function
`OFILL`	Uses fill characters for delay
`OFDEL`	Fill is `DEL`, else `NULL`

`NLDLY`	Selects newline delays:
	`NL0`
	`NL1`

`CRDLY`	Selects carriage-return delays:
	`CR0`
	`CR1`
	`CR2`
	`CR3`

`TABDLY`	Selects horizontal tab delays or tab expansion:
	`TAB0`
	`TAB1`
	`TAB2`

`TAB3`	Expands tabs to spaces.

`BSDLY`	Selects backspace delays:
	`BS0`
	`BS1`

`VTDLY`	Selects vertical tab delays:
	`VT0`
	`VT1`

`FFDLY`	Selects form feed delays:
	`FF0`
	`FF1`

The initial output control value is `OPOST`, `ONLCR`, and `TAB3`. If set, the bits have the following meanings:

`OPOST` Postprocesses output characters as indicated by the remaining flags; if `OPOST` is not set, characters are transmitted without change.

`OLCUC` Transmits a lowercase alphabetic character as the corresponding uppercase character. This function is often used in conjunction with `IUCLC`.

`ONLCR` Transmits the `NL` character as the `CR-NL` character pair.

`OCRNL` Transmits the `CR` character as the `NL` character.

`ONOCR` Does not transmit a `CR` character when at column 0 (first position).

`ONLRET` Performs the `CR` function. The `NL` character is assumed to do the carriage-return function; the column pointer is set to 0, and the delays specified for `CR` are used. Otherwise, the `NL` character is assumed to do just the line-feed function; the column pointer remains unchanged. If the `CR` character is actually transmitted, the column pointer also is set to 0.

The following delay bits specify the length of time the 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.

`OFILL` Transmits fill characters for a delay instead of a timed delay. This is useful for terminals that have a high baud rate that need only a minimal delay.

`OFDEL` Sets the fill character to `DEL` (`NULL` by default).

`NLDLY` Selects newline delays. Newline delay lasts for 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 are transmitted.

`CRDLY` Selects carriage-return delays. 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.

`TABDLY` Selects horizontal tab delays. Horizontal-tab delay type 1 depends on the current column position; type 2 is about 0.10 seconds, and type 3 specifies that tabs will be expanded into spaces. If `OFILL` is set, two fill characters are transmitted for any delay.

`TAB3` Expands tabs to spaces.

`BSDLY` Selects backspace delays. Backspace delay lasts for about 0.05 seconds. If `OFILL` is set, one fill character is transmitted.

`VTDLY` Selects vertical tab delays. Vertical-tab delay lasts for about 2 seconds. If `OFILL` is set, two fill characters are transmitted.

`FFDLY` Selects form-feed delays. Form-feed delay lasts for about 2 seconds. If `OFILL` is set, two fill characters are transmitted.

The actual delays depend on line speed and system load.

Control Modes

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

`CBAUD`	Baud rate:	
	`B0`	Hang up
	`B50`	50 Bd
	`B75`	75 Bd
	`B110`	110 Bd
	`B134`	134 Bd
	`B150`	150 Bd
	`B200`	200 Bd
	`B300`	300 Bd
	`B600`	600 Bd
	`B1200`	1200 Bd
	`B1800`	1800 Bd
	`B2400`	2400 Bd
	`B4800`	4800 Bd
	`B9600`	9600 Bd
	`B19200`	19200 Bd
	`EXTA`	External A
	`B38400`	38400 Bd `EXTB` External B
`CSIZE`	Character size:	
	`CS5`	5 bits
	`CS6`	6 bits
	`CS7`	7 bits
	`CS8`	8 bits
`CSTOPB`	Sends 2 stop bits, else 1	
`CREAD`	Enables receiver	
`PARENB`	Enables parity	
`PARODD`	Odd parity, else even	
`HUPCL`	Hangs up on last close	

CLOCAL Local line, else dial-up

The initial hardware control value after an open(2) system call is B9600, CS8, CREAD, and HUPCL.

If set, the bits in the c_cflag field have the following meanings:

CBAUD Specifies the baud rate. The 0 Bd rate, B0, hangs up the connection. If B0 is specified, the data-terminal-ready signal is not asserted. Usually, this disconnects the line. For any particular hardware, impossible speed changes are ignored. The default is B9600, which specifies a 9600 Bd rate.

CSIZE Specifies the character size (in bits) for both transmission and reception. This size does not include the parity bit if any. The default is CS8, which specifies a character size of 8 bits.

CSTOPB Specifies the number of stop bits used. If CSTOPB is set, 2 stop bits are used; otherwise, 1 stop bit is used (for example, at 110 Bd, 2 stops bits are required).

CREAD Enables the receiver. If CREAD is not set, no characters are received. CREAD is set by default.

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

PARODD Specifies odd parity if PARENB is set.

HUPCL Disconnects the line when the last process with that line open closes it or terminates; that is, the data-terminal-ready signal is not asserted. HUPCL is set by default.

CLOCAL If CLOCAL is set, the line is assumed to be a local, direct connection with no modem control. If CLOCAL is not set, modem control is assumed.

Local Modes

The c_lflag field of the argument structure is used by the line discipline to control terminal functions. The basic line discipline provides the following:

ISIG Enables signals

ICANON Enables canonical input (erase and kill processing)

XCASE Enables canonical upper/lower presentation

ECHO Enables echo

ECHOE Echoes erase character as BS-SP-BS

ECHOK Echoes NL after kill character

ECHONL Echoes NL

NOFLSH Disables flush after interrupt or quit

TOSTOP Sends SIGTTOU for background output

IEXTEN Enables extended (implementation-defined) functions

The initial line-discipline control value is ISIG, ICANON, ECHO, and ECHOK.

If set, the bits have the following meanings:

ISIG Enables signals. Each input character is checked against the special control characters INTR, QUIT, SWTCH, SUSP, STATUS, and DSUSP. If an input character matches one of these control characters, the function associated with that character is performed. If ISIG is not set, checking is not done. Thus, these special input functions are possible only when ISIG is set. To disable these functions individually, change the value of the control character to an unlikely or impossible value (for example, 0377).

ICANON Enables canonical processing. This enables the erase and kill edit functions, and the assembly of input characters into lines delimited by NL, EOF, EOL, and EOL2. If ICANON is not set, read requests are satisfied directly from the input queue. A read is not satisfied until at least MIN characters have been received, or the time-out 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 is represented in tenths of seconds.

XCASE Specifies canonical presentation of uppercase and lowercase characters. If XCASE is set and ICANON also is set, an uppercase letter is accepted on input when prefaced by a \ character, and an uppercase letter is prefaced by a \ character on output. In this mode, the following escape sequences are generated on output and accepted on input:

For:	Use:
`	\´
		\!
~	\^
{	\(
}	\)
\	\\

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

ECHO Enables echo. If ECHO is set, characters are echoed as received.

ECHOE Echoes the erase character as ASCII BS SP BS.

ECHOK Echoes an NL character after the kill character.

ECHONL Echoes an NL character.

NOFLSH Disables the normal flush of the input and output queues associated with the interrupt (INTR), quit (QUIT), and suspend (SUSP) characters. This bit should be set when restarting system calls that read from or write to a terminal; see sigaction(2).

TOSTOP Sends the SIGTTOU signal for background output. If a process tries to write to its controlling terminal when it is not in the foreground process group for that terminal, the signal is sent. This signal usually stops the process. Otherwise, the output generated by that process is output to the current output stream. Processes that are blocking or ignoring SIGTTOU signals are excepted and allowed to produce output, if any.

IEXTEN Enables the following extended (implementation-defined) functions: special characters (WERASE, REPRINT, DISCARD, and LNEXT) and local flags (TOSTOP, ECHOCTL, ECHOPRT, ECHOKE, FLUSHO, and PENDIN).

When ICANON is set, the following echo functions are possible:

- If ECHO and ECHOE are set, the erase character (ERASE and WERASE) is echoed as ASCII BS SP BS, which clears the last character from a terminal.
- If ECHOK is set and ECHOKE is not set, the NL character is echoed after the kill character to emphasize that the line is deleted. An escape character (\) or an LNEXT character that precedes the erase or kill character removes any special function.
- If ECHONL is set, the NL character is echoed even if ECHO is not set. This is useful for terminals set to local echo (half-duplex).

Minimum and Time-out

The MIN and TIME values are described in the Noncanonical Mode Input Processing subsection. The initial value of MIN is 1, and the initial value of TIME is 0.

Terminal Size

The number of lines and columns on the terminal's display is specified in the winsize structure defined by sys/termios.h, which includes the following members:

```
unsigned  short   ws_row;     /* rows, in characters        */
unsigned  short   ws_col;     /* columns, in characters     */
unsigned  short   ws_xpixel;  /* horizontal size, in pixels */
unsigned  short   ws_ypixel;  /* vertical size, in pixels   */
```

termio Structure

Some ioctl(2) requests use the termio structure. It is defined by the sys/termio.h include file and includes the following members:

```
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 characters */
```

The `c_cc` array defines the special control characters. The symbolic name `NCC` is the size of the control-character array and also is defined by the `sys/termio.h` include file. The relative positions, subscript names, and typical default values for each function are as follows:

0	`VINTR`	`DEL`
1	`VQUIT`	`FS`
2	`VERASE`	`#`
3	`VKILL`	`@`
4	`VEOF`	`EOT`
5	`VEOL`	`NUL`
6	`VEOL2`	`NUL`
7	reserved	

The calls that use the `termio` structure affect only the flags and control characters that can be stored in the `termio` structure; all other flags and control characters are unaffected.

Supported `ioctl` Requests

This subsection lists the primary `ioctl`(2) requests supported by devices and STREAMS modules providing the `termios` interface (see `terminal`(3C)). All devices or modules may not support some `ioctl`(2) requests. The functionality provided by these requests also is available through the preferred function call interface specified on the `terminal`(3C) man page.

The following `ioctl` requests are supported:

`TCFLSH` Flushes input and/or output queues. If *arg* is 0, `TCFLSH` flushes the input queue; if *arg* is 1, it flushes the output queue; if *arg* is 2, it flushes both the input and output queues.

`TCGETA` Gets the parameters associated with the terminal and stores them in the `termio` structure referenced by *arg*.

`TCSBRK` Waits for the output to drain. If *arg* is 0, `TCSBRK` sends a break (0 bits for 0.25 seconds).

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

`TCSETAF` Waits for the output queue to empty, then flushes the input queue and sets the new parameters from the `termio` structure referenced by *arg*.

`TCSETAW` Waits for the output queue to empty before setting the new parameters from the `termio` structure referenced by *arg*. When changing parameters that affect output, use this request.

`TCXONC` Enables start and stop control. If *arg* is 0, `TCXONC` suspends output; if *arg* is 1, it restarts suspended output; if *arg* is 2, it suspends input; and if *arg* is 3, it restarts suspended input.

`TIOCGWINSZ` Stores the terminal size in the `winsize` structure to which *arg* points.

`TIOCSWINSZ` Sets the terminal size from the `winsize` structure to which *arg* points. If the new sizes are different from the old sizes, a `SIGWINCH` signal is set to the process group of the terminal.

`TCCLRCTTY` Clears the controlling tty connection.

`TCSETCTTY`	Defines the terminal as the controlling tty for a session.
`TCGETPGRP`	Gets the foreground process group ID for the session.
`TCSETPGRP`	Sets the foreground process group ID for the session.
`TCSIG`	Interrupts all outstanding asynchronous I/O in the foreground process group, and it sends the process group a signal.
`TCGETDEV`	Gets the device number of the terminal.
`TCTTRD`	Reserved for use by the SCP Interactive facility.
`TCRDFL`	Enables the "master read failure with user reads" option.

FILES

`/dev/*`

`/dev/tty*`

`/usr/include/termio.h`

SEE ALSO

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

`fork`(2), `ioctl`(2), `setpgrp`(2), `setsid`(2), `sigaction`(2), `signal`(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

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

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

NAME

tpddem – Tape daemon interface

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The tape pseudo device driver provides user-level control over the tape devices. That is, the tape daemon (TPD) interface. The tpddem interface provides a mechanism for the tape daemon to control the tape devices. The daemon handles allocation and scheduling of tape devices. Only a process with a sysadm category may have direct access to a TPD device.

The tpddem interface contains hooks at critical points such as open operations, the first I/O operation, error recovery, and close operations. The tape daemon uses these hooks to control user access to the tape devices, including validation of open operations, automatic volume switching, label processing, and error recovery. When the tape daemon activates one of these hooks, the tpddem interface suspends the current user process and sends a signal to the tape daemon. When the controlling process completes, it sends an ioctl(2) system call to the tpddem driver to resume the user process. The controlling process may resume the user process that has an error.

The hooks in the tape device drivers cause the tape daemon process that has the TPD device open to preempt the user process that is using a tape device. If no process has TPD open, the tape driver falls through these hooks.

The tpddem interface supports the close(2), ioctl(2), and open(2) system calls. The open(2) system call activates hooks in the TPD driver for the calling process. The super user must run the calling process; the device is exclusive (only one process may have the device open). The close(2) system call deactivates the hooks in the TPD driver. The ioctl(2) system call sends commands to the tape driver.

The tpddem structure that follows, as defined in the sys/tpddem.h include file, is used to communicate between the driver and the controlling process.

```
/* TPD daemon control table */

struct tpddem {
       int     flags;              /* open/close flags            */
       int     owner;              /* ID of controlling process   */
       int     pid;                /* process ID of tape demon    */
       struct  proc *p;            /* proc address of controlling */
                                   /* process                     */
       long    demdev;             /* tpddev device (major, minor) */
       long    reserved0;          /* reserved for future use     */
       long    reserved1;          /* reserved for future use     */
       long    reserved2;          /* reserved for future use     */
       long    reserved3;          /* reserved for future use     */
       struct  bdtab tab[TPD_MAXBMXDV]; /* substructure 1 per device */
};

 /* TPD device control/status structure */

 struct bdtab {
        word  name;    /* Name of device                           */
        int   dev;     /* Minor device number                      */
        int   flag;    /* Device requires daemon process           */
        int   func;    /* Current tpd device function              */
        int   user;    /* User ID                                  */
        long  status;  /* Current device status                    */
        int   error;   /* Error return to user                     */
        int   reslct;  /* Device in reselect                       */
        int   wait;    /* User sleeping on this table              */
        int   rval;    /* Function-dependent return value          */

 /* Following used for reselect to new device */

        int  newdev;   /* Ordinal of new device table              */

 /* For tape positioning and user end-of-volume communication      */

        int   dmn_int1
        int   dmn_int2
        int   dmn_int3
        int   dmn_int4
        int   partition;      /* Partition to position to          */
        int   filesec;        /* File section to position to       */
        int   datablock;      /* Datablock to position to          */
        int   absaddr;        /* Absolute address to position to   */
```

```
        long  dmn_tpvdev;     /* tpd device number                 */
        long  oflags;         /* open flags                        */
        long  dmn_reserved1;  /* reserved for future use           */
        long  dmn_reserved2;  /* reserved for future use           */
        long  dmn_reserved3;  /* reserved for future use           */
};
```

The available `ioctl` requests, as defined in the `sys/tpddem.h` include file, are as follows:

`TDM_BFMON` Buffer monitor.

`TDM_CUEOV` Clears user end-of-volume (EOV) flag.

`TDM_FIRST` First daemon function.

`TDM_GET` Returns the structure `tpddem` to the passed-in address.

`TDM_OBIT` Returns a list of job IDs for jobs that have terminated and are still recorded in the system reservation table.

`TDM_RLS` Removes the passed-in job ID from the system reservation table.

`TDM_RSL` Reselects to a new device.

`TDM_RSM` Resumes the user process for the device specified by the passed-in structure `bdtab`. The `error` member of `bdtab` is set into the user's error status. This request must follow the `TDM_SET` request.

`TDM_RSV` Saves the process group ID passed in the system reservation table. When the last member of a process group exits and it is recorded in this table, a signal is sent to the controlling process for resource control.

`TDM_SET` Sets the fields in the `bdtab` system structure from the passed-in structure `bdtab`. Also clears the `flag` member of `bdtab`. This request must precede the `TDM_RSM` request.

`TDM_SUEOV` Sets user EOV flag.

`TDM_WDN` Waits for the specified device to complete its current operation.

The tape daemon reason codes, which are used to signal the tape daemon, are as follows:

`TDR_ABN` `abn` flag set.

`TDR_CLOSE` User `close` flag.

`TDR_DEMPROC` Tape daemon processing flag.

`TDR_IO` First I/O request.

`TDR_OPEN` User `open` flag.

`TDR_RDONLY` Request to write on a read-only file.

`TDR_REASON` First reason code flag. All reason codes must be greater than this value.

TDR_USRREQ	User function.
TDM_WTM	Write tape mark flag.

FILES

/dev/bxmdem	Tape daemon (TPD) interface
/usr/include/sys/tpddem.h	Structure definition of tpddem

SEE ALSO

tape(4)

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

Tape Subsystem User's Guide, Cray Research publication SG–2051

NAME

`tty` – Controlling terminal interface

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `/dev/tty` file is a synonym for the control terminal, if any, associated with the process group of each process. It is useful for programs or shell sequences that want to be sure of writing messages on the terminal no matter how output has been redirected. When output to the terminal is desired, you also can use `/dev/tty` for programs that require the name of a file for output. In this way, the program does not have to find out the terminal that is currently in use.

FILES

`/dev/tty`

SEE ALSO

`pty`(4), `termio`(4), `tp`(4) (CRAY Y-MP systems)

NAME

vme – VME (FEI-3) network interface

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The VME (FEI-3) network interface is a channel-to-VME backplane adapter that connects a Cray Research system with a VMEbus based, front-end computer.

The special files for the FEI-3 interface are in the `/dev` directory.

FEI-3 special file names have the following naming convention:

`/dev/vme`*nn*

nn Minor device number ("logical path" in IOS terminology) for the VME interface

Support for the FEI-3 is provided through the IOS as if the FEI-3 were an NSC adapter. The device is otherwise treated as an NSC adapter; for more information, see `hy`(4).

FILES

`/dev/vme*`

`/usr/include/sys/hy.h`

`/usr/include/sys/hysys.h`

SEE ALSO

`hy`(4)

`ioctl`(2), `listio`(2), `read`(2), `reada`(2), `write`(2), `writea`(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

NAME

`xdd` – Physical disk device interface

IMPLEMENTATION

CRAY J90se systems

CRAY T90 systems

DESCRIPTION

The files in `/dev/xdd` are special files that allow read and write operations to physical disk devicesconnected to the MPN-1 (SCSI disks), the FCN-1 (Fibre SCSI disks), or the HPN-1/HPN-2 (HIPPI disks). Each file represents one slice of a physical disk device. The files in `/dev/xdd` are character special files that may be used directly to read and write physical disk slices. Usually, they are called to perform I/O on behalf of higher-level logical disk device drivers. For I/O on a character disk device, read and write operations must transfer multiples of the physical device sector size and all seek operations must be on physical sector size boundaries.

The files in `/dev/xdd` are not usually mountable as file systems, although you may combine one or more physical disk slices to make a mountable logical disk device (see `dsk`(4), `ldd`(4), and `mount`(8)).

The files in `/dev/xdd` are created by using the `mknod` command (see `mknod`(8)). Each must have a unique minor device number, along with other parameters used to define a physical disk slice.

The `mknod`(8) command for physical disk devices is as follows:

`mknod` *name type major minor dtype iopath start length flags altpath unit* [*ifield*]

name Descriptive file name for the device (for example, `xdd/scr0230`).

type Type of the device data being transferred. Devices in `/dev/xdd` are character devices denoted by a `c`.

major Major device number for physical disk devices. The `dev_xdd` name label in the `/usr/src/uts/c1/cf/devsw.c` file denotes the major device number for physical disk devices. You can specify the major number as `dev_xdd`.

minor Minor device number for this slice.

dtype Physical disk device type defition, consisting of 32 bits defined as follows:

Bit 0 - 11 Defines the sector size.

Bit 12 - 19 Defines the type field. Currently this field is not used.

Mode bit Mode bit. If mode is 1, the sector size given is in words per sector. If mode is 0 (the default value), the sector size given is *iouni ts*. An *iounit* is 512 words. The mode bit is not currently supported.

iopath Specifies the GigaRing number on which the device is located, the node number to which the disk is connected, and the controller slot number (ION channel number) of the device.

Bit 0 - 2 Defines the controller slot number.

Bit 3 - 8 Defines the ION node number.

Bit 9 - 15 Defines the Ring number.

For example: An *iopath* of 0110204(octal) indicates Controller Number 4, ION node number 2, and Ring Number 011(octal) or 9(decimal).

For disk devices connected to the MPN-1 (SCSI disks), the SCSI controller slot number is in the range 0 through 8. For disk devices connected to the FCN-1 (Fibre SCSI disks), the controller number is in the range 0 through 4.

For HIPPI disks, the *iopath* is represented by an octal number in the format 0*rrrnnc* where:

rrr The GigaRing number

nn The node number of the HPN

c The channel on the HPN

start Absolute starting sector number of the slice.

length Number of blocks (sectors) in the slice.

flags Flags for physical disk device control. They are mainly used for diagnostic and maintenance purposes. Usually, the flags field should be 0 for slices in `/dev/xdd`. For HIPPI disks, the *flags* field is 0.

```
#define S_CONTROL       001    /* control device                 */
#define S_NOBBF         002    /* no bad block forwarding        */
#define S_NOERREC       004    /* no error recovery              */
#define S_NOLOG         010    /* no error logging               */
#define S_NOWRITEB      020    /* no write behind                */
#define S_CWE           040    /* control device write enable    */
#define S_NOSORT        0200   /* no disk sort                   */
#define S_NODEVINT      0400   /* no device intimate functions   */
#define S_MODE_MASK     0777   /* Mask containing special flags */
```

altpath Optional alternate *iopath* that you can use as a back-up path to the physical disk device's second port.

unit The disk device unit number for device types that support multiple units on the same channel.

The unit number is defined as follows:

Bit 0 - 7 Disk unit number

Bit 8 - 15 Logical Unit Number

For HIPPI disks, this is the disk's facility number.

ifield (HIPPI disks only). The hardware address of the HIPPI disk in the HIPPI network.

FILES

`/dev/xdd/*`

`/usr/include/sys/xdd.h`

`/usr/src/c1/io/xdd.c`

SEE ALSO

`dsk`(4), `ldd`(4), `mdd`(4), `qdd`(4), `sdd`(4),

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

NAME

intro – Introduction to TCP/IP networking facilities and files

SYNOPSIS

```
#include <sys/socket.h>
#include <net/route.h>
#include <net/if.h>
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

This section describes the TCP/IP networking facilities, protocols, and data files available in UNICOS. Unless otherwise noted, all include (header) files mentioned in this section are in the /usr/include directory.

Protocols

All network protocols are associated with a protocol family. A *protocol family* provides the services that allow the protocol implementation 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, although the current protocol implementations do not. A protocol family usually is composed of several protocols. The current system fully supports the DARPA Internet protocol family. Raw socket interfaces are provided to the Internet Protocol (IP) and Internet Control Message Protocol (ICMP) layer of the DARPA Internet. For a description of the user protocols of the Internet protocol family, see inet(4P); the protocols are further detailed in tcp(4P) and udp(4P). For a description of Internet Protocol (IP) and Internet Control Message Protocol (ICMP), see the icmp(4P) and ip(4P) man pages.

A network interface is similar to a device interface. *Network interfaces* compose the lowest layer of the networking subsystem, mapping the network system to the device drivers. Associated with a protocol family is an *address format*. An interface may support more than one protocol family or address format. Protocols generally accept only one type of address format, usually determined by the addressing structure inherent in the design of the protocol family and network architecture. TCP/IP uses the following address format:

```
#define  AF_INET  2     /* internetwork: UDP, TCP, etc. */
```

The network facilities provide limited packet routing. A *routing table* is a table in the UNICOS kernel composed of a set of data structures; it is used to select the appropriate network interface when transmitting packets. This table contains one entry for each route to a specific network or host. A user process, called the *routing daemon* (gated(8)), maintains this database with the aid of a routing socket (see route(4P)). Only the super user may perform routing table manipulations.

A routing table entry has the following format, as defined in the `net/route.h` include file:

```
struct rtentry {
        struct  radix_node rt_nodes[2]; /* tree glue, and other values */
#define      rt_key(r)       ((struct sockaddr *)((r)->rt_nodes->rn_key))
#define      rt_mask(r)      ((struct sockaddr *)((r)->rt_nodes->rn_mask))
        struct  sockaddr *rt_gateway;   /* value */
#ifndef _CRAY
        short   rt_flags;               /* up/down?, host/net */
#else
        int     rt_flags;               /* up/down?, host/net */
#endif
        short   rt_refcnt;              /* # held references */
        u_long  rt_use;                 /* raw # packets forwarded */
        struct  ifnet *rt_ifp;          /* the answer: interface to use */
        struct  ifaddr *rt_ifa;         /* the answer: interface to use */
        struct  sockaddr *rt_genmask;   /* for generation of cloned routes */
        caddr_t rt_llinfo;              /* pointer to link level info cache */
        struct  rt_metrics rt_rmx;      /* metrics used by rx'ing protocols */
        short   rt_idle;                /* easy to tell llayer still live */
#ifdef _CRAY
#define NRTGID 32                       /* maximum size of the gid list */
        long    rt_gid[NRTGID];         /* list of gids for restricting */
        u_long  rt_admmtu;              /* Administrator mtu for the route */
        int     rt_time;                /* Time when the route was added */
        long    rt_pathmtu;             /* Est. minimum MTU over path */
        int     rt_pmtuchanged;         /* Timer for last change to pathmtu */
                                        /* 0 means it has never been changed */
      /*
         * Yes, these are IP specific.  When it becomes necessary to
         * break up rentries according to AF, we'll do something.
         */
        char    rt_iptos;               /* 8-bit IP TOS field */

        /* Handle group id list as a variable length array */
        int      rt_gidcnt;             /* number of gid's in gidlist */
        long    *rt_gidlist;            /* gid list */

#endif /* _CRAY */
};
```

The rt_flags variable is defined as follows:

```
#define RTF_UP          0x1         /* route usable                      */
#define RTF_GATEWAY     0x2         /* destination is a gateway          */
#define RTF_HOST        0x4         /* host entry (net otherwise)        */
#define RTF_REJECT      0x8         /* host or net unreachable           */
#define RTF_DYNAMIC     0x10        /* created dynamically (by redirect) */
#define RTF_MODIFIED    0x20        /* modified dynamically (by redirect)*/
#define RTF_DONE        0x40        /* message confirmed                 */
#define RTF_MASK        0x80        /* subnet mask present               */
#define RTF_CLONING     0x100       /* generate new routes on use        */
#define RTF_LLINFO      0x400       /* generated by ARP or ESIS          */
#define RTF_STATIC      0x800       /* manually added                    */
#define RTF_NOFORWARD   0x1000      /* do not forward through            */
#define RTF_EXCLGID     0x2000      /* gid list is exclusive             */
#define RTF_PROTO2      0x4000      /* protocol-specific routing flag    */
#define RTF_PROTO1      0x8000      /* protocol-specific routing flag    */
#define RTF_TOSMATCH    0x10000     /* high-level match required for TOS */
#define RTF_NOMTUDISC   0x40000     /* don’t do path MTU discovery       */
```

Three types of entries are in a routing table: the route for a host, the route for all hosts on a network, and the route for any destination not matched by entries of the first two types (a wildcard route). When you boot the system, each network interface that has been configured automatically installs a routing table entry when it wants to have packets sent through it. Typically, the interface specifies the route through a direct connection to the destination host or network. If the route is direct (that is, not through a gateway), the transport layer of a protocol family usually requests that the packet be sent to the host specified in the packet. Otherwise, the request may be to address the packet to a host different from the eventual recipient (that is, the packet is forwarded).

Routing table entries installed by a user process may not specify the hash field (rt_hash), reference count field (rt_refcnt), use field (rt_use), interface field (rt_ifp), time when the route was added (rt_time), discovered path MTU field (rt_pathmtu), or timer for the last change to the path MTU (rt_pmtuchanged); the routing routines fill these in.

If a route is in use when it is deleted (that is, rt_refcnt has a nonzero value), the resources associated with it are not reclaimed until further references to it are released.

The routing code returns EEXIST if requested to duplicate an existing entry, ESRCH if requested to delete a nonexistent entry, or ENOBUFS if insufficient resources were available to install a new route.

User processes may read the routing tables through a routing socket or the /dev/mem special file (see mem(4)).

The rt_use field contains the number of packets sent along the route.

The system administrator may use the rt_mtu field to specify a maximum transmission unit size for connections established over the route.

A wildcard routing entry is specified with a destination address value of `INADDR_ANY`. (The symbol `INADDR_ANY` is defined in the `netinet/in.h` include file as 0.) Wildcard routes are used only when the system does not find a route to the destination host and network. The combination of wildcard routes and routing redirects can provide an economical mechanism for routing traffic.

Addressing

An address format is associated with each protocol family. All network addresses adhere to a general structure, called a *sockaddr*. However, each protocol imposes finer and more specific structure, generally renaming the variant.

```
struct sockaddr {
      u_char  sa_len:32;
      u_char  sa_family:32;
      char    sa_data[32];
};
```

The `sa_len` field contains the total length of the structure, which may exceed 16 bytes. The following address values for `sa_family` are known to 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. */
```

Interfaces

Each network interface in a system corresponds to a path through which messages may be sent and received. The TCP/IP network interfaces supported on UNICOS are the loopback interface (`lo`(4)), the HYPERchannel adapter interface (`hy`(4)), the VME network interface (`vme`(4)), the HSX channel interface (`hsx`(4)), and the HIPPI interface (`hippi`(4)). A network interface usually has a hardware device associated with it, although certain interfaces (such as `lo`) do not.

`ioctl`(2) Requests

You also can use the following `ioctl`(2) requests to manipulate network interfaces. Unless specified, the request takes an `ifreq` structure as its parameter. This structure is defined in the `net/if.h` include file, as follows:

```
struct ifreq {
#define      IFNAMSIZ     16
      char   ifr_name[IFNAMSIZ];       /* if name, for example, "en0" */
      union {
             struct sockaddr ifru_addr;
             struct sockaddr ifru_dstaddr;
             struct sockaddr ifru_broadaddr;
             short  ifru_flags;
             int    ifru_metric;
             caddr_t      ifru_data;
             struct {
                    mac_label_t ifru_minlabel;
                    mac_label_t ifru_maxlabel;
             } ifru_seclabel;
             long ifru_auth;
      } ifr_ifru;
#define      ifr_addr        ifr_ifru.ifru_addr     /* address */
#define      ifr_dstaddr     ifr_ifru.ifru_dstaddr  /* other end of p-to-p link */
#define      ifr_broadaddr   ifr_ifru.ifru_broadaddr/* broadcast address */
#define      ifr_flags       ifr_ifru.ifru_flags    /* flags */
#define      ifr_metric      ifr_ifru.ifru_metric   /* metric */
#define      ifr_data        ifr_ifru.ifru_data     /* for use by interface */
#ifdef _CRAY
#define ifr_mtu              ifr_ifru.ifru_metric   /* if mtu */
#define ifr_rbufs            ifr_ifru.ifru_metric   /* if read buffers */
#define ifr_wbufs            ifr_ifru.ifru_metric   /* if write buffers */

#define      ifr_minlabel    ifr_ifru.ifru_seclabel.ifru_minlabel /* minimum sec level */
#define      ifr_maxlabel    ifr_ifru.ifru_seclabel.ifru_maxlabel  /* maximum sec level */
#define      ifr_auth        ifr_ifru.ifru_auth            /* valid authorities */
#endif /* _CRAY */
};
```

A list of the available `ioctl`(2) requests follows:

SIOCGIFADDR	Gets the interface address.
SIOCGIFBRDADDR	Gets the broadcast address for protocol family and interface.
SIOCGIFCONF	Gets the interface configuration list. This request takes an `ifconf` structure as a value-result parameter. Initially, you should set the `ifc_len` field of this structure to the size of the buffer to which `ifc_buf` points. On return, it contains the length, in bytes, of the configuration list.
SIOCGIFDSTADDR	Gets the point-to-point address for the interface.
SIOCGIFFLAGS	Gets the interface flags.
SIOCGIFLABEL	Gets the interface security label.

SIOCGIFMETRIC	Gets interface metric.
SIOCGIFMTU	Gets interface maximum transmission unit size.
SIOCGIFNETMASK	Gets interface subnet mask.
SIOCGIFRBUFS	Gets count of read buffers posted to low-level driver.
SIOCGIFWBUFS	Gets maximum count of write buffers that may be posted to low-level driver.
SIOCSIFADDR	Sets the interface address. Following the address assignment, the initialization routine for the interface is called.
SIOCSIFBRDADDR	Sets the broadcast address for protocol family and interface.
SIOCSIFDSTADDR	Sets the point-to-point address for the interface.
SIOCSIFFLAGS	Sets the interface flags field. If the interface is marked as being down, any processes currently routing packets through the interface are notified.
SIOCSIFLABEL	Sets the interface security label.
SIOCSIFMETRIC	Sets interface routing metric. Only user-level routers use the metric.
SIOCSIFMTU	Sets interface maximum transmission unit size.
SIOCSIFRBUFS	Sets count of read buffers to post to low-level driver.
SIOCSIFWBUFS	Sets maximum count of write buffers that may be posted to low-level driver.

The `ifconf` structure is defined in `net/if.h`, as follows:

```
/*
 * Structure used in SIOCGIFCONF request.
 * Used to retrieve interface configuration for machine
 * (useful for programs that must know all networks accessible).
 */

struct  ifconf {
        int     ifc_len;                /* size of associated buffer    */
        union  {
                caddr_t ifcu_buf;
                struct  ifreq *ifcu_req;
        } ifc_ifcu;
#define ifc_buf ifc_ifcu.ifcu_buf       /* buffer address               */
#define ifc_req    ifc_ifcu.ifcu_req     /* array of structures returned */
} ;
```

Network Data Files

TCP/IP commands (described in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011, and the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022) and TCP/IP library functions (described in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080) use network data files.

A list of the network data files follows.

Data file	Man page entry	Description
`ftpusers`	`ftpusers`(5)	List of unacceptable `ftp`(1B) users
`hosts`, `hosts.bin`	`hosts`(5)	Contains network host name database
`hosts.equiv`	`hosts.equiv`(5)	Public information for validating remote autologin
`.netrc`	`netrc`(5)	TCP/IP autologin information for outbound `ftp` requests
`networks`	`networks`(5)	Network name database
`protocols`	`protocols`(5)	Protocol name database
`.rhosts`	`rhosts`(5)	List of trusted remote hosts and account names
`services`	`services`(5)	Network service name database

FILES

`/usr/include/net/if.h`	Include file for `ifreq` structure
`/usr/include/net/route.h`	Kernel packet forwarding database
`/usr/include/sys/socket.h`	Include file that defines address families

SEE ALSO

`hippi`(4), `hsx`(4), `hy`(4), `icmp`(4P), `inet`(4P), `ip`(4P), `mem`(4), `route`(4P), `tcp`(4P), `udp`(4P), `vme`(4), `ftpusers`(5), `hosts`(5), `hosts.equiv`(5), `netrc`(5), `networks`(5), `protocols`(5), `rhosts`(5), `services`(5)

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

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

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

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

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

"Internet Transport Protocols," XSIS 028112, Xerox System Integration Standard

NAME

arp – Address Resolution Protocol

SYNOPSIS

pseudo-device ether

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The arp protocol dynamically maps between DARPA Internet and 10-Mbyte/s Ethernet addresses. The 10-Mbyte/s Ethernet and FDDI interface drivers use arp. It is not specific to the Internet protocols, to FDDI, or to 10-Mbyte/s Ethernet, but this implementation currently supports only Ethernet and FDDI.

The arp protocol caches Internet-Ethernet address mappings. When an interface requests a mapping for an address not in the cache, arp queues the message, which requires the mapping, and broadcasts a message on the associated network that requested the address mapping. If a response is provided, the new mapping is cached and any pending message is transmitted. arp queues at most one packet while waiting for a response to a mapping request; only the most recently transmitted packet is kept. If the target host does not respond after several requests, the host is considered to be down for a short period (normally about 20 seconds), allowing an error to be returned to transmission attempts during this interval. The error is EHOSTDOWN for a non-responding destination host, and EHOSTUNREACH for a non-responding router.

The arp cache is stored in the system routing table as dynamically-created host routes. The route to a directly-attached broadcast network is installed as a "cloning" route (one with the RTF_CLONING flag set), causing routes to individual hosts on that network to be created on demand. These routes time out periodically (normally 20 mintues after validation); entries are not validated when not in use. An entry for a host which is not responding is a "reject" route (one with the RTF_REJECT flag set).

arp entries may be added, deleted, or changed with the arp(8) utility. Manually-added entries may be temporary or permanent, and may be "published," in which case the system will respond to ARP requests for that host as if it were the target of the request. In the past, ARP was used to negotiate the use of a trailer encapsulation. This is no longer supported.

The arp protocol watches passively for hosts impersonating the local host (that is, a host that responds to an ARP mapping request for the local host’s address).

MESSAGES

The following message indicates that arp has discovered another host on the local network that responds to mapping requests for its own Internet address:

```
duplicate IP address!! sent from ethernet address: %x:%x:%x:%x:%x:%x
```

BUGS

ARP packets on the Ethernet use only 42 bytes of data; however, the smallest legal Ethernet packet is 60 bytes (not including the cyclic redundancy code (CRC)). Some systems may not enforce the minimum packet size.

SEE ALSO

`route`(4P) in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

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

`inet`(4P) for a description of Internet protocol family

`arp`(8) to display address resolution display and control

`ifconfig`(8) to configure network interface parameters in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

An Ethernet Address Resolution Protocol, RFC 826, Dave Plummer, Network Information Center, SRI

NAME

`icmp` – Internet Control Message Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_RAW, IPPROTO_ICMP);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The Internet Control Message Protocol (ICMP) is a mechanism that hosts and gateways in an IP internetwork use to exchange error messages and other maintenance information. The ICMP is defined in *DARPA Internet Request for Comments*, RFC 792. UNICOS provides a limited interface to ICMP for user programs; sending incorrect ICMP information can cause problems throughout the internetwork.

Transmission

ICMP is a datagram protocol; therefore, a raw ICMP socket has no connections (`listen`(2) and `accept`(2) return errors). The `sendto`(2) system call is the normal method for transmission through an ICMP socket. The `connect`(2) system call, though not supported by the underlying protocol, permanently associates the socket with a destination for future transmissions, thus enabling use of the `send`(2) and `write`(2) system calls on the socket.

The entire contents of a transmission (the buffer in a `sendto`(2) system call, `send`(2) system call, or `write`(2) system call) is packaged into the data portion of one datagram for transmission. To be accepted at the destination, the buffer to be transmitted must contain a valid ICMP datagram, beginning with an ICMP header with a valid checksum. UNICOS does not enforce this locally; datagrams that are not valid appear to be sent with no problems, but they confuse or are ignored by their recipients. An ICMP datagram structure is defined in the `netinet/ip_icmp.h` include file.

Reception

The `recvfrom`(2) system call is the normal method for receiving data through an ICMP socket. The `connect`(2) system call, though not supported by the underlying protocol, permanently associates the socket with a source of future transmissions, thus enabling use of the `recv`(2) and `read`(2) system calls on the socket.

Received ICMP datagrams are presented to the user from the socket with their ICMP header included. The ICMP datagram structure is defined in `netinet/icmp.h` as the `icmp` structure.

Only ICMP datagrams that have correct data checksums are passed to user programs. Datagrams that have incorrect checksums (which may have been corrupted in transit) are discarded without notice. The following four kinds of ICMP datagrams are not available to user programs:

- Echo request (type 8)
- Time-stamp request (type 13)
- Information request (type 15)
- Address mask request (type 17)

These ICMP datagrams are handled in the kernel and then discarded.

If you do not provide enough buffer space for the entire available datagram in a call to `recvfrom`(2), `read`(2), or `recv`(2), excess bytes at the end of the datagram will be silently discarded.

NOTES

Only the super user may create an ICMP socket.

ICMP provides no mechanism for out-of-band data.

You cannot specify IP options for an outbound ICMP datagram.

MESSAGES

A socket operation may fail with one of the following errors returned:

`EADDRNOTAVAIL`
: Returned if the process tried to create a socket with a network address for which no network interface exists.

`EISCONN`
: Returned if the socket already has a connection when a connection is tried, or if the process is trying to send a datagram with the destination address specified when the socket is already connected.

`ENOBUFS`
: Returned if the system ran out of memory for an internal data structure.

`ENOTCONN`
: Returned if the process is trying to send a datagram, but no destination address has been specified and the socket is not already connected.

FILES

`/usr/include/netinet/in.h`	Include file for Internet addresses
`/usr/include/netinet/ip_icmp.h`	Defines ICMP datagram structure
`/usr/include/sys/socket.h`	Include file that defines address families

SEE ALSO

`inet`(4P), `intro`(4P), `ip`(4P)

`accept`(2), `connect`(2), `listen`(2), `read`(2), `recv`(2), `send`(2), `write`(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

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

DARPA Internet Request for Comments, RFC 792

NAME

igmp – Internet Group Management Protocol

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The Internet Group Management Protocol (IGMP) is a mechanism that hosts and routers on the physical network use to identify which hosts currently belong to which multicast groups. Multicast routers use this information to determine which multicast diagrams to forward on to potential interfaces. The IGMP is defined in DARPA Internet Request for Comments, RFC 1112.

IGMP is considered part of the Internet Protocol (IP) layer, and messages are transmitted in IP datagrams.

SEE ALSO

inet(4P), intro(4P), ip(4P)

NAME

inet – Description of Internet protocol family

SYNOPSIS

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

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The Internet protocol family is a collection of protocols, piled on top of the Internet Protocol (IP) layer, that use the Internet address format. The Internet protocol family is composed of the IP itself, the Internet Control Message Protocol (ICMP), the Transmission Control Protocol (TCP), and the user datagram protocol (UDP).

The Internet protocol family provides protocol support for the socket types SOCK_STREAM, SOCK_DGRAM, and SOCK_RAW. TCP supports the SOCK_STREAM abstraction, UDP supports the SOCK_DGRAM abstraction, and the SOCK_RAW socket type provides a raw interface to IP and ICMP.

Internet addresses are 4-byte quantities stored in network standard format. The netinet/in.h include file defines an Internet address, as follows:

```
struct  {
      u_long       st_addr:32;
} s_da;
#define s_addr    s_da.st_addr
```

Sockets bound to the Internet protocol family use the following addressing structure:

```
struct  sockaddr_in {
      u_long sin_len:32;
      u_long sin_family:32;
      _SHORTPAD
      u_short sin_port;
      struct      in_addr  sin_addr;
      char  sin_zero[16];
} ;
```

You may create sockets by using address INADDR_ANY to cause wildcard matching on incoming messages.

FILES

`/usr/include/netinet/in.h`	Include file for Internet addresses
`/usr/include/sys/types.h`	Include file for socket types

SEE ALSO

`icmp`(4P), `ip`(4P), `tcp`(4P), `udp`(4P)

NAME

ip – Description of Internet Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>
s = socket(AF_INET, SOCK_RAW, protocol);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The Internet Protocol (IP) is the network layer protocol that the Internet protocol family uses. IP provides the functions necessary to deliver an Internet datagram from a source to a destination over a TCP/IP network. IP is defined in MIL-STD 1777 and in RFC 791. You probably will not have to use IP sockets unless you are developing new upper-layer protocols. The Transmission Control Protocol (TCP) and user datagram protocol (UDP) are for more general use. (For more information, see `tcp`(4P) and `udp`(4P), respectively.)

Transmission

IP is a datagram protocol, so a raw IP socket has no connections; that is, `listen`(2) and `accept`(2) return errors. The `sendto`(2) system call is the normal method for transmission through an IP socket. The `connect`(2) system call, though not supported by the underlying protocol, permanently associates the socket with a destination for future transmissions, thus enabling use of the `send`(2) and the `write`(2) system calls on the socket.

The entire contents of a transmission (the buffer in a `sendto`(2) system call, `send`(2) system call, or `write`(2) system call) is packaged into the data portion of one datagram for transmission. The kernel provides the IP header. The protocol number in the IP header is the protocol number specified in the `socket`(2) system call used to create the socket.

Reception

The `recvfrom`(2) system call is the normal method for receiving data through an IP socket. The `connect`(2) system call, though not supported by the underlying protocol, permanently associates the socket with a destination for future transmissions, thus enabling use of the `recv`(2) and `read`(2) system calls on the socket.

Received IP datagrams are presented to the user from the socket with the IP header included. The IP header structure is defined in the `netinet/ip.h` include file as the `ip` structure. The datagram contents immediately follow the header in the receiving buffer. Because IP neither generates nor tests checksums on the data, the data may be garbled in transmission.

Only incoming datagrams that carry the protocol number specified in the socket(2) system call are available from an IP socket.

If you do not provide enough buffer space for the entire available datagram in a call to recvfrom(2), read(2), or recv(2), excess bytes at the end of the datagram will be discarded without notice. To determine whether bytes are missing, check the number of bytes returned from recvfrom(2) against the ip_len field in the header of the received datagram.

Options

You may set options at the IP level when using higher-level protocols that are based on IP (such as TCP and UDP). You also may access the IP protocol through a raw socket when developing higher-level protocols or developing special-purpose applications. The following are IP-level setsockopt(2) and getsockopt(2) system call options:

IP_OPTIONS

Provides IP options to be transmitted in the IP header of each outgoing packet or examines the header options of incoming packets. You may use IP options with any socket type in the Internet family. The IP protocol specification (RFC 791) specifies the format of IP options to be sent, except the list of addresses for Source Route options must include the first-hop gateway at the beginning of the list of gateways. The first-hop gateway address is extracted from the option list and the size adjusted accordingly before use. To disable previously specified options, use a zero-length buffer. An example of this option follows:

```
setsockopt (s, IPPROTO_IP, IP_OPTIONS, NULL, 0);
```

IP_RECVDSTADDR

Causes the recv(2) system call to return the destination IP address for a UDP datagram. This option is enabled only on a SOCK_DGRAM socket.

IP_TOS, IP_TTL

Sets the type-of-service (TOS) and time-to-live (TTL) fields in the IP header for SOCK_STREAM and SOCK_DGRAM sockets. The following example includes both these parameters:

```
int tos = IPTOS_LOWDELAY;  /* see netinet/in.h */
setsockopt (s, IPPROTO_IP, IP_TOS, &tos, sizeof(tos));

int ttl = 60;              /* max = 255 */
setsockopt (s, IPPROTO_IP, IP_TTL, &ttl, sizeof(ttl));
```

Multicast Options

IP multicasting is supported only on AF_INET sockets of type SOCK_DGRAM and SOCK_RAW and only on networks in which the interface driver supports multicasting. The following are multicast options:

IP_ADD_MEMBERSHIP

Places a host in a multicast group. A host must become a member of a multicast group before it can receive datagrams sent to the group. An example follows:

```
struct ip_mreq mreq;
setsockopt (s, IPPROTO_IP, IP_ADD_MEMBERSHIP, &mreq, sizeof(mreq));
```

In the previous example, `mreq` has the following structure:

```
struct ip_mreq {
     struct in_addr imr_multiaddr;  /* multicast group to join */
     struct in_addr imr_interface;  /* interface to join on */
}
```

The `imr_interface` parameter should be `INADDR_ANY` to choose the default multicast interface, or the IP address of a particular multicast-capable interface if the host is multihomed. Membership is associated with one interface; programs that run on multihomed hosts may have to join the same group on more than one interface. You may add up to the specified value in the `IP_MAX_MEMBERSHIPS` option on one socket.

`IP_DROP_MEMBERSHIP`
Drops a membership in a multicast group. The `mreq` parameter contains the same values as those used to add the membership. An example follows:

```
struct ip_mreq mreq;
setsockopt (s, IPPROTO_IP, IP_DROP_MEMBERSHIP, &mreq, sizeof(mreq));+
```

When the socket is closed or the process exits, memberships also are dropped.

`IP_MULTICAST_IF`
Specifies the interface on which each multicast transmission is sent. The following is an example of this option:

```
struct in_addr addr;
setsockopt (s, IPPROTO_IP, IP_MULTICAST_IF, &addr, sizeof(addr));
```

In this example, you can use `addr` to specify the local address of the desired interface, or `INADDR_ANY` to specify the default interface. To obtain the local IP address and multicast capability of an interface, use the `ioctl`(2) system calls `SIOCGIFCONF` and `SIOCGIFFLAGS`. Most applications do not have to use this option.

`IP_MULTICAST_LOOP`
Gives the sender explicit control over whether subsequent datagrams are looped back. The following is an example of this option:

```
u_char loop;          /* 0 = disable, 1 = enable (default) */
setsockopt (s, IPPROTO-IP, IP_MULTICAST_LOOP, &loop, sizeof(loop));
```

If a multicast datagram is sent to a group to which the sending host itself belongs (on the outgoing interface), a copy of the datagram is, by default, looped back by the IP layer for local delivery. This option improves performance for applications that may have no more than one instance on a single host (such as a router daemon) by eliminating the overhead of receiving their own transmissions. Generally, applications for which more than one instance may be on a single host (such as a conferencing program) or for which the sender does not belong to the destination group (such as a time-querying program) should not use this option.

A multicast datagram sent with an inital TTL greater than 1 may be delivered to the sending host on a different interface from that on which it was sent if the host belongs to the destination group on that other interface. The loopback control option does not affect such delivery.

`IP_MULTICAST_TTL`

Changes the TTL for outgoing multicast datagrams to control the scope of the multicasts. The following example uses this option:

```
u_char ttl;         /* range:  0 to 255, default = 1 */
setsockopt (s, IPPROTO_IP, IP_MULTICAST_TTL, &ttl, sizeof(ttl));
```

Datagrams with a TTL of 1 are not forwarded beyond the local network. Multicast datagrams with a TTL of 0 are not transmitted on any network, but they may be delivered locally if the sending host belongs to the destination group and if multicast loopback has not been disabled on the sending socket. If a multicast router is attached to the local network, multicast datagrams with a TTL greater than 1 may be forwarded to other networks.

Raw IP Options

Raw IP sockets are connectionless, and they usually are used with the `sendto`(2) and `recvfrom`(2) system calls. The `connect`(2) system call also may fix the destination for future packets, in which case, you may use the `read`(2) or `recv`(2), and `write`(2) or `send`(2) system calls.

If the *protocol* argument is 0, the default protocol `IPPROTO_RAW` is used for outgoing packets, and only incoming packets destined for that protocol number are received. If `proto` is a nonzero value, that value is used on outgoing packets and is used to filter incoming packets.

Outgoing packets automatically have an IP header prepended to them (based on the destination address and the protocol number with which the socket is created), unless the `IP_HDRINCL` option has been set. Incoming packets are received with header and options intact. A raw IP option follows:

`IP_HDRINCL`

Indicates the complete IP header is included with the data. You may use this option only with the `SOCK_RAW` type. An example follows:

```
#include <netinet/ip.h>
int hincl = 1;
setsockopt (s, IPPROTO_IP, IP_HDRINCL, &hincl, sizeof(hincl));
```

Unlike previous UNICOS releases, the program must set all of the fields of the IP header, including the following:

```
ip->ip_v = IPVERSION;
ip->ip_hl = hlen >> 2;
ip->ip_id = 0;  /* 1 = on, 0 = off */
ip->ip_off = offset;
```

If the header source address is set to the kernel, the program chooses an appropriate address.

NOTES

Only the super user may create an IP socket.

IP provides no mechanism for out-of-band data.

MESSAGES

If a socket operation fails, it returns one of the following error messages:

`EADDRNOTAVAIL`
: The process tried to create a socket by using a network address for which no network interface exists.

`EISCONN`
: The socket already has a connection when a connection is tried, or the process is trying to send a datagram with the destination address specified when the socket is already connected.

`ENOBUFS`
: The system ran out of memory for an internal data structure.

`ENOTCONN`
: The process is trying to send a datagram, but no destination address has been specified and the socket is not already connected.

When you are setting or getting IP options, the following errors specific to IP can occur:

`[EINVAL]`
: An unknown socket option name was specified.

`[EINVAL]`
: An unknown socket descriptor was specified.

`[EINVAL]`
: The IP option field was improperly formed (for example, an option field was shorter than the minimum value or longer than the option buffer provided).

FILES

`/usr/include/netinet/in.h`	Include file for Internet addresses
`/usr/include/netinet/ip.h`	Defines the IP header structure
`/usr/include/sys/socket.h`	Defines address families

SEE ALSO

`icmp`(4P), `igmp`(4P), `inet`(4P), `intro`(4P), `tcp`(4P), `udp`(4P)

`accept`(2), `connect`(2), `getsockopt`(2), `listen`(2), `read`(2), `recv`(2), `recvfrom`(2), `send`(2), `socket`(2), `write`(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

RFC 791

NAME

nfs – UNICOS network file system (UNICOS NFS)

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The UNICOS network file system (UNICOS NFS) is an implementation of the network file system (NFS) for Cray Research systems running UNICOS. NFS was originally designed and developed to reduce the need for local disk storage in distributed environments. NFS executes on a wide variety of computing hardware and is implemented on operating systems other than its development home in the Berkeley UNIX environment.

NFS allows file systems to be shared across a network of machines. The standard system calls for file operations (close(2), open(2), read(2), and write(2)) system calls are used to access both local and network-based files; the location of files can be transparent to the user. Standard permission mechanisms are used to control file access.

The user interface to NFS is transparent; that is, there is no user interface. After a system is configured, users simply read and write their files, whether they are on local disk or exist elsewhere on the network.

Client and Server Modes

Client NFS allows users to make standard system calls (close(2), create(2), delete(3C), open(2), read(2), and write(2)) to a portion of file name space on which a network file system has been mounted (by using mount(8)). These calls are intercepted by a process in the client system, translated into the corresponding NFS requests, and packaged for transmission across a network to a server machine. With UNICOS NFS, the file system switch (FSS) facilitates mapping between local requests and NFS requests.

Server NFS implementations allow a portion of their local file system to be exported (made available for remote mounting). When NFS requests for exported file systems are received, the server performs the indicated operation; about 20 file system operations are supported. In the case of read or write requests, the indicated data is returned to the remote NFS client (for a read operation), or written to local disk (for a write operation).

Client and server implementations are logically separate. Some implementations (for example, the implementation of NFS for MS-DOS) are client only; that is, they can make use of remote systems but cannot export file systems of their own. Other implementations (for example, the implementation of NFS for VMS) are server only; they can respond to requests from client systems but do not allow remote file systems to be mounted on their local namespace. UNICOS NFS includes both client and server modes.

A server can grant access to a specific file system to certain clients by adding an entry for that file system to the server's /etc/exports file (see exports(5)). A client gains access to that file system by using the mount(2) system call, which requests a file handle for the file system itself. After the client mounts the file system, the server issues a file handle to the client for each file (or directory) the client accesses. If the file is somehow removed on the server side, the file handle becomes stale (disassociated with a known file).

A client cannot export file systems that it has mounted over the network; therefore, clients must mount file systems directly from the server on which the file systems reside. The user ID (UID) and group ID (GID) mappings must be the same between client and server; however, the server maps UID 0 (the super user) to UID-2 before performing access checks for a client. This inhibits super-user privileges on remote file systems.

Network Interface (RPC, XDR, and UDP/IP)

NFS is a set of high-level protocols, based on a remote procedure call (RPC) model; NFS network requests are made through calls to remote procedures that implement NFS file system semantics. (The libraries that contain these RPC procedures have been available since the UNICOS 2.0 release.)

RPC makes use of an intermediate data representation for all information sent to and received from the network. This intermediate form is called *External data representation* (XDR). (The libraries that contain these XDR procedures have been available since the UNICOS 2.0 release.)

Stateless Servers

Within NFS, all state information (such as open file status) is maintained by the client implementations, while the servers are said to be *stateless*. Servers are thus relatively simple, and recovery operations are more reliable than with *stateful* servers.

Mount and Lock Managers

Managers handle operations that are related to particular implementations of file systems, such as UNIX or UNICOS file systems, but are not deemed to be universal operations. Currently, two managers are defined for NFS: one handles the mount protocol; the other handles file locking. The managers typically run as user-level processes, and they communicate with the kernel implementation in very carefully defined ways.

NOTES

UNICOS NFS is a licensed product that also requires UNICOS and UNICOS TCP/IP licenses. Therefore, UNICOS NFS may not be available at your site.

In most NFS implementations, user ID (UID) and group ID (GID) values are the same between client and server. The network information service (NIS) distributed data lookup service is often used to manage `passwd`(5) and `group`(5) files to ensure consistency across an entire NFS domain. For more information about using the network information service (NIS) feature, see the *UNICOS Networking Facilities Administrator's Guide*, Cray Research publication SG–2304.

UNICOS NFS sites also can use the ID mapping facility, which provides for the operation of UNICOS NFS in environments that are not administratively homogeneous. For more information, see the *UNICOS Networking Facilities Administrator's Guide*, Cray Research publication SG–2304. Typically, the server maps UID 0 (root) to UID-2 before performing access checks for a client.

MESSAGES

Generally, physical disk I/O errors detected on the server are returned to the client for action. If the server is down or inaccessible, the client sees the following console message:

```
NFS:  file server not responding: still trying.
```

The client continues to send the request until it receives an acknowledgment from the server. This means that the server can crash (or power down) and come back up without any special action required by the client. It also means that the client process requesting the I/O will block and remain insensitive to signals, sleeping inside the kernel at priority `PRIBIO`.

SEE ALSO

`exports`(5), `fstab`(5)

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

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

`mount`(8), `nfsaddmap`(8), `nfsclear`(8), `nfsd`(8), `nfslist`(8), `nfsmerge`(8), `nfsstat`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

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

NAME

`route` – Kernel packet forwarding database

SYNOPSIS

```
#include <sys/socket.h>
#include <net/if.h>
#include <net/route.h>

int family
s = socket(PF_ROUTE, SOCK_RAW, family);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `route` facility performs packet routing services. The kernel maintains a routing information database known as a *routing table*, which selects the appropriate network interface when packets are transmitted. A user process (or possibly multiple cooperating processes) maintains this routing table by sending messages over a special kind of socket. The use of this routing table supersedes the use of the fixed-size `ioctl`, implemented in earlier releases. Only the super user can make changes to the routing table.

The operating system might spontaneously emit routing messages in response to external events, such as receipt of a redirect, or failure to locate a suitable route for a request.

Routing table entries can exist for a specific host or for all hosts on a generic subnetwork (as specified by a bit mask and value under the mask). To achieve the effect of wildcard or default routes, use a mask of all 0's. Some routes might be hierarchical.

When the system is booted and addresses are assigned to the network interfaces, each protocol family installs a routing table entry for each interface when it is ready for traffic. Usually, the protocol specifies the route through each interface as a direct connection to the destination host or network. If the route is specified as direct, the transport layer of a protocol family usually requests that the packet be sent to the same host specified in the packet. Otherwise, it requests the interface to address the packet to the gateway listed in the routing entry (that is, the packet is forwarded).

When the kernel is routing a packet, it first tries to find a route to the destination host. Failing that, it makes a search for a route to the network of the destination. Finally, it chooses any route to a default (or wildcard) gateway. If no entry is found, the destination is declared to be unreachable, and if any processes are listening for messages on the routing socket, a routing-miss message is generated.

A wildcard routing entry is specified with a destination address value of 0. Wildcard routes are used only when the system does not find a route to the destination host and network. The combination of wildcard routes and routing redirects can provide an economical mechanism for routing traffic.

You can open the channel for passing routing control messages by using the socket call shown in the SYNOPSIS section. You can designate the *family* argument to be AF_UNSPEC, which provides routing information for all address families, or you can restrict it to a specific address family by designating a specific *family* argument. You can have more than one routing socket open per system.

Messages are formed with a header followed by a small number of socket addresses (sockaddr fields), interpreted by position, and delimited by the length entry in the sockaddr field (this length is variable).

A bit mask within the header specifies which address is present; the position sequence is least-significant to most-significant bit within the vector. The kernel returns any messages it receives, and copies are sent to all interested listeners. The kernel provides the process ID for the sender; the sender can use an additional sequence field to distinguish between outstanding messages. However, when kernel buffers are exhausted, message replies might be lost.

The kernel can reject certain messages; it indicates rejection by filling in the rtm_errno field. The routing code returns EEXIST if requested to duplicate an existing entry, ESRCH if requested to delete a nonexistent entry, or ENOBUFS if insufficient resources were available to install a new route. In the current implementation, all routing processes run locally; the values for rtm_errno are available through the normal errno mechanism, even if the routing reply message is lost.

A process can avoid the expense of reading replies to its own messages by issuing a setsockopt() call (see getsockopt(2)), indicating that the SO_USELOOPBACK option at the SOL_SOCKET level will be turned off. A process can ignore all messages from the routing socket by issuing a shutdown(2) system call for further input.

If a route is in use when it is deleted, the routing entry is marked down and removed from the routing table, but the resources associated with it are not reclaimed until all references to it are released. User processes can obtain information about the routing entry to a specific destination by using a RTM_GET message, or by calling the sysctl routine.

Messages

Following is a list of the messages and their meanings that the routing facility generates:

```
#define RTM_ADD      0x1 /* Add route */
#define RTM_DELETE   0x2 /* Delete route */
#define RTM_CHANGE   0x3 /* Change metrics, flags, or gateway */
#define RTM_GET      0x4 /* Report information */
#define RTM_LOSING   0x5 /* Kernel suspects partitioning */
#define RTM_REDIRECT 0x6 /* Told to use different route */
#define RTM_MISS     0x7 /* Lookup failed on this address */
#define RTM_RESOLVE  0xb /* Request to resolve dst to LL addr */
#define RTM_LOCK     0x8 /* Lock metric values */
```

Message Headers

An example of a message header follows:

```
struct rt_msghdr {
        u_short rmt_msglen;          /* Skip nonunderstood messages */
        u_char  rtm_version;         /* Future binary compatibility */
        u_char  rtm_type;            /* Message type */
        u_short rmt_index;           /* Index for associated ifp  */
        pid_t   rmt_pid;             /* Identify sender */
        int     rtm_addrs;           /* Bit mask for sockaddrs in msg */
        int     rtm_seq;             /* For sender to identify action */
        rtm_errno;                   /* Why failed
        int     rtm_flags;           /* Kernel and message flags */
        int     rtm_use;             /* From rtentry */
        u_long  rtm_inits;           /* Values to be initialized */
        struct  rt_metrics rtm_rmx; /* Metrics themselves */
};
```

Metrics Structure

The structure for the metrics is as follows:

```
struct rt_metrics {
        u_long rmx_locks;     /* Kernel must leave these values alone */
        u_long rmx_mtu;       /* MTU for this path */
        u_long rmx_hopcount;  /* Max hops expected */
        u_long rmx_expire;    /* Lifetime for route ( e.g., redirect) */
        u_long rmx_recvpipe;  /* Inbound delay-bandwith product */
        u_long rmx_sendpipe;  /* Outbound delay-bandwith product */
        u_long rmx_ssthresh;  /* Outbound gateway buffer limit */
        u_long rmx_rtt;       /* Estimated round-trip time */
        u_long rmx_rttvar;    /* Estimated rtt variance */
};
```

Flags

Flags include the following values:

```
#define RTF_UP        0x1     /* Route usable */
#define RTF_GATEWAY   0x2     /* Destination is a gateway */
#define RTF_HOST      0x4     /* Host entry (net otherwise) */
#define RTF_REJECT    0x8     /* Host or net unreachable */
#define RTF_DYNAMIC   0x10    /* Created dynamically (by redirect) */
#define RTF_MODIFIED  0x20    /* Modified dynamically (by redirect) */
#define RTF_DONE      0x40    /* Message confirmed */
#define RTF_MASK      0x80    /* Subnet mask present */
#define RTF_CLONING   0x100   /* Generate new routes on use */
#define RTF_LLINFO    0x400   /* Generated by ARP or ESIS */
#define RTF_STATIC    0x800   /* Manually added */
#define RTF_NOFORWARD 0x1000  /* Do not forward through */
#define RTF_EXCLGID   0x2000  /* gid list is exclusive */
#define RTF_PROTO2    0x4000  /* Protocol-specific routing flag */
#define RTF_PROTO1    0x8000  /* Protocol-specific routing flag */
#define RTF_TOSMATCH  0x10000 /* High-level match required for TOS */
#define RTF_NOMTUDISC 0x40000 /* Do not do path MTU discovery */
```

Metric Value Specifiers

Specifiers for metric values in `rmx_locks` and `rtm_inits` are as follows:

```
#define RTV_MTU       0x1   /* Initialize or lock _mtu */
#define RTV_HOPCOUNT  0x2   /* Initialize or lock _hopcount */
#define RTV_EXPIRE    0x4   /* Initialize or lock _hopcount */
#define RTV_RPIPE     0x8   /* Initialize or lock _recvpipe */
#define RTV_SPIPE     0x10  /* Initialize or lock _sendpipe */
#define RTV_SSTHRESH  0x20  /* Initialize or lock _ssthresh */
#define RTV_RTT       0x40  /* Initialize or lock _rtt */
#define RTV_RTTVAR    0x80  /* Initialize or lock _rttvar */
```

Address Specifiers

Specifiers for which addresses are present in the messages are as follows:

```
#define RTA_DST      0x1    /* Destination sockaddr present */
#define RTA_GATEWAY  0x2    /* Gateway sockaddr present */
#define RTA_NETMASK  0x4    /* Netmask sockaddr present */
#define RTA_GENMASK  0x8    /* Cloning mask sockaddr present */
#define RTA_IFP      0x10   /* Interface name sockaddr present */
#define RTA_IFA      0x20   /* Interface addr sockaddr present */
#define RTA_AUTHOR   0x40   /* sockaddr for author of redirect */
```

SEE ALSO

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

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

NAME

`tcp` – Transmission Control Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_STREAM, 0);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The Transmission Control Protocol (TCP) provides reliable, flow-controlled, two-way byte streams between pairs of programs running on hosts in an Internet Protocol (IP) network. The protocol is defined in *DARPA Internet Request for Comments*, RFC 793.

TCP allows for multiple byte streams between a pair of hosts by associating each connection on a host with a port number. The Internet addresses of the connected hosts and two port numbers, one on each host, uniquely identifies a connection. Port numbers are integers, specified in the `bind`(2) or `connect`(2) system call, ranging in value from 1 to 65,535. By Internet convention, some ports are reserved for well-known services (for example, hosts provide TELNET service by listening for connections on port 23). These services are listed in *DARPA Internet Request for Comments*, RFC 1010. By Berkeley UNIX convention, only the super user may bind to ports that have numbers lower than 1024.

Connection

Sockets that use the TCP protocol are either active or passive. *Active sockets* initiate connections to passive sockets. TCP sockets are created as active sockets; to create a passive socket, you must have the `listen`(2) system call after the socket is bound to a port that has the `bind`(2) system call. Only passive sockets may use the `accept`(2) system call to accept incoming connections. Only active sockets may use the `connect`(2) system call to initiate connections. After a socket has been made passive, it cannot be made active again.

Passive sockets may ‘‘underspecify’’ their location to match incoming connection requests from multiple networks. This technique, termed *wildcard addressing*, allows one server to provide service to clients on multiple networks. To create a socket that listens on all networks, bind the socket to the Internet address `INADDR_ANY` (defined in the `netinet/in.h` include file). The TCP port may still be specified at this time; if you do not specify the port, the system will assign one.

Transmission

TCP is a byte-stream protocol with connections; `write`(2) is the usual method of sending on a TCP socket. The `send`(2) system call is useful for sending out-of-band data. The `sendto`(2) system call (see `send`(2)) works, but it is misleading because the destination address is ignored.

TCP is a byte stream, not a word stream. Although you can send data types other than bytes over a TCP connection, for example, by passing the address of an integer to `write`(2), no guarantee exists that an integer will come out at the other end. The receiving host may have a different word size, a different byte order, or other incompatibilities.

Reception

The `read`(2) system call is the usual method of receiving on a TCP socket. The `recv`(2) system call is useful for receiving out-of-band data. The `recvfrom`(2) system call (see `recv`(2)) works, but it is misleading because the source address is ignored.

Because TCP presents data from a socket in arbitrary chunks as the data becomes available from the Internetwork, TCP sockets are more likely than regular files to return fewer bytes than requested. Be sure to check the return value from `read`(2).

Disconnection

Executing the `shutdown`(2) system call on a TCP socket before the `close`(2) system call allows you to close down one side of the connection.

Options

The `SO_KEEPALIVE` option (defined in the `sys/socket.h` include file) causes the code in the kernel that handles TCP protocol periodically to send packets that contain no data; these packets are acknowledged and discarded. This tests whether the data path to the other end of the connection is open. If the other end fails to respond to these keep-alive packets, the connection will be closed and the next socket operation will return `ETIMEDOUT`.

Other options have their socket-level effects; the socket-level effect is explained in the `getsockopt`(2) system call.

You can use options at the IP transport level with TCP (see `ip`(4P).

TCP supports several socket options that you can set by using `setsockopt`(2) and test by using `getsockopt`(2). The option level for the `setsockopt`(2) call is the protocol number for TCP, available from `getprotobyname`(3C).

Most socket-level options take an `int` type value. For `setsockopt`(2), the value must be nonzero to enable a Boolean option, or 0 to indicate that the option will be disabled. You can use the following options with TCP:

Option	**Description**
`TCP_MAXSEG`	Gets maximum segment size. You cannot set this option.
`TCP_NODELAY`	Toggles the `no delay` flag. Under most circumstances, TCP sends data when it is presented. When outstanding data has not yet been acknowledged, it gathers small amounts of output to be sent in one packet after an acknowledgment has been received. For a few clients (such as window systems that send a stream of mouse events that receive no replies), this packetization might cause significant delays. TCP provides this Boolean option to defeat this algorithm.

`TCP_WINSHIFT` Sets the TCP window shift count. You must set this option on a socket before the use of the `connect`(2) or `accept`(2) system call. A value of –1 turns off window shift; a value of 0 through 14 turns on window shift with the requested window size. `getsockopt`(2) returns up to 24 bytes (the `TR_SENDWNDSHIFT` value in the first word, the send window shift value in the second word, and the receive window shift value in the third word).

NOTES

Only the super user may bind a socket to a port number lower than 1024.

MESSAGES

A socket operation may fail with one of the following values in `errno`:

`EADDRINUSE`
An attempt is made to create a socket by using a port that has already been allocated.

`EADDRNOTAVAIL`
The process tried to create a socket with a network address for which no network interface exists.

`ECONNREFUSED`
The remote peer actively refuses connection establishment (usually because no process is listening to the port).

`ECONNRESET`
The remote peer forces the connection to be closed.

`EINVAL`
An option value or socket that is not valid was specified for the `setsockopt`(2) system call.

`EISCONN`
The socket already has a connection when a connection is tried on the `connect`(2) system call, or you cannot use the `setsockopt`(2) `TCP_WINSHIFT` option on an established connection.

`ENOBUFS`
The system ran out of memory for an internal data structure.

`ETIMEDOUT`
A connection was dropped because of excessive retransmissions.

FILES

`/usr/include/netinet/in.h`	Include file for Internet addresses
`/usr/include/netinet/tcp.h`	Include file for TCP addresses
`/usr/include/sys/socket.h`	Address family definition

SEE ALSO

inet(4P), intro(4P), ip(4P)

accept(2), bind(2), close(2), connect(2), getsockopt(2), listen(2), read(2), recv(2), setsockopt(2), shutdown(2), write(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

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

DARPA Internet Request for Comments, RFC 793 and RFC 1010

NAME

udp – Internet User Datagram Protocol

SYNOPSIS

```
#include <sys/socket.h>
#include <netinet/in.h>

s = socket(AF_INET, SOCK_DGRAM, 0);
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The Internet User Datagram Protocol (UDP) is a simple datagram protocol that `tftp`(1B) and the `rpc`(3C) library routines use. The protocol is defined in *DARPA Internet Request for Comments*, RFC 768.

UDP allows for multiple endpoints on a host by associating each endpoint with a port number. Port numbers are integers, specified by the `bind`(2) or `connect`(2) system call, ranging in value from 1 to 65,535. By Internet convention, some ports are reserved for well-known services, listed in *DARPA Internet Request for Comments*, RFC 1010; for example, hosts provide TFTP (see `tftp`(1B)) service by listening for datagrams on port 69. By Berkeley UNIX convention, only the super user may listen on ports with numbers fewer than 1024.

Transmission

UDP is a datagram protocol, therefore, a UDP socket has no connections; that is, the `listen`(2) and `accept`(2) system call return errors. The `sendto`(2) system call (see `send`(2)) is the normal method for transmission through a UDP socket. The `connect`(2) system call, though not supported by the underlying protocol, permanently associates the socket with a destination for future transmissions, thus enabling use of the `send`(2) system call and the `write`(2) system call on the socket.

The entire contents of a transmission (the buffer in a `sendto`(2), `send`(2), or `write`(2) system call) are packaged into the data portion of one datagram for transmission. The kernel provides the UDP and Internet Protocol (IP) headers. You can configure the kernel to calculate the UDP data checksum; if not, UDP packets will go out without protection against transmission errors.

Reception

The `recvfrom`(2) system call (see `recv`(2)) is the normal method for receiving data through a UDP socket. The `connect`(2) system call, though not supported by the underlying protocol, permanently associates the socket with a destination for future transmissions, thus enabling use of the `recv`(2) system call and the `read`(2) system call on the socket.

The kernel checks the UDP data checksum in arriving datagrams and discards garbled datagrams without presenting them to the user through the socket.

If you do not provide enough buffer space for the entire available datagram in a call to `recvfrom`(2), `read`(2), or `recv`(2), excess bytes at the end of the datagram will be discarded without notice.

MESSAGES

A socket operation may fail with one of the following errors returned:

`EADDRINUSE`
The process tries to create a socket by using a port that has already been allocated.

`EADDRNOTAVAIL`
The process tried to create a socket with a network address for which no network interface exists.

`EISCONN`
The socket already has a connection when a connection is tried, or the process is trying to send a datagram with the destination address specified when the socket is already connected.

`ENOBUFS`
The system ran out of memory for an internal data structure.

`ENOTCONN`
The process is trying to send a datagram, but no destination address has been specified and the socket is not already connected.

FILES

`/usr/include/netinet/in.h`	Include file for Internet addresses
`/usr/include/netinet/udp.h`	UDP header file
`/usr/include/sys/socket.h`	Address family definition

SEE ALSO

`inet`(4P), `intro`(4P), `tcp`(4P)

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

`accept`(2), `bind`(2), `connect`(2), `listen`(2), `read`(2), `recv`(2), `send`(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

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

DARPA Internet Request for Comments, RFC 768 and RFC 1010

NAME

intro – Introduction to file formats

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

Section 5 outlines the formats of certain UNICOS files. These files include header files, data files, and output files from UNICOS utilities. The files in this section fall into one of three categories:

- User category
- Administrator category
- Analyst category

Within each of these categories, there are two subdivisions; files a user, administrator, or kernel process can change, and files used as a reference, template, or data file.

The following three tables detail the division of the entries in section 5. Some entries fall into more than one category; these are entries that a user references, but an administrator changes. For example, the group(5) entry describes the /etc/group file, which can be viewed by a user and changed by an administrator.

Many entries in this section describe *header files*. Header files are files in a specific format that more than one program (such as compilers, assemblers, and system utilities) use, often for data interchange between programs. You must enter the names of header files in the predefined format that is shown on the man page. When applicable, the C struct declarations for the file formats are given. In this manual, header files are referred to as *include files* because they usually are found in the /usr/include or /usr/include/sys directory. In the DESCRIPTION section of the entries, the full path name for a header file is given only when it is not in either of these directories.

User Category

The man pages in this category describe entries of interest to UNICOS users. An * symbol marks files that users set up or modify.

Man page	Description	File(s)
aliases	Define alias database for sendmail(8)	/usr/lib/aliases
a.out	Loader output file	/usr/include/a.out.h
ar	Archive file format	/usr/include/ar.h
bld	Relocatable library files format	/usr/src/cmd/bld/bld.h
cpio	cpio(1) archive file format	
cshrc	C shell start-up and termination files	.cshrc*, .login*, .logout*
def_seg	Loader directives files	/lib/segdirs/def_seg*
exrc	Start-up files for ex(1) and vi(1)	.exrc*

Man page	Description	File(s)
fcntl	File control options	/usr/include/fcntl.h
group	Group-information file format	/etc/group
mailrc	Start-up files for mailx(1)	.mailrc*
motd	File that contains message of the day	/etc/motd
netrc	TCP/IP autologin information file for outbound ftp(1B) requests	$HOME/.netrc*
nl_types	Defines message system variables	nl_types.h
passwd	Password file format	/etc/passwd
profile	Format of Posix shell start-up file	.profile*
publickey	Public key database	/etc/publickey*
relo	Relocatable object table format under UNICOS	/usr/include/relo.h
rhosts	List of trusted remote hosts and account names	.rhosts*
sccsfile	Source Code Control System (SCCS) file format	s.*file*
symbol	UNICOS symbol table entry format	/usr/include/symbol.h
tapetrace	Tape daemon trace file format	/usr/spool/tape/trace.daemon /usr/spool/tape/trace.bmx*xxx* /usr/include/tapereq.h
taskcom	Task common table format	
types	Definition of primitive system data types	/usr/include/sys/types.h
updaters	Configuration file for NIS updating	/etc/yp/updaters*
uuencode	Encoded uuencode file format	
values	Machine-dependent values definition file	/usr/include/values.h

Administrator Category

The man pages in this category describe files of interest to an administrator. An * symbol marks files that the administrator modifies.

Man page	Description	File(s)
acid	Account ID information file format	/etc/acid
acl	User access control lists format	/usr/include/sys/acl.h
aft	ASCII flaw table	/etc/aft/*
confval	Configuration file for various products	
cshrc	C shell start-up and termination files	/etc/cshrc*
dump	Incremental file system dump format	/usr/src/cmd/fs/dump
exports	Directories to export to NFS clients	/etc/exports*

Man page	Description	File(s)
fslrec	File system error log record format	/dev/fslog /usr/include/sys/fslog.h /usr/include/sys/fslrec.h /usr/include/sys/types.h
fstab	File that contains static information about file systems	/etc/fstab*
ftpusers	List of unacceptable ftp(1B) users	/etc/ftpusers*
gated-config	Gated configuration file syntax	/etc/gated.conf
gettydefs	Speed and terminal settings used by getty(8)	/etc/gettydefs
group	Group-information file format	/etc/group*
hosts	TCP/IP host name database	/etc/hosts*
hosts.equiv	Public information for validating remote autologin	/etc/hosts.equiv*
inetd.conf	Internet super-server configuration file	/etc/inetd.conf*
inittab	Script for init process	/etc/inittab*
iptos	IP Type-of-Service database	/etc/iptos
issue	Login message file	/etc/issue*
krb.conf	Kerberos configuration file	
krb.realms	Host to Kerberos realm translation file	
ldesc	Logical disk descriptor file	/usr/include/sys/ldesc.h
mailrc	Start-up files for mailx(1)	/usr/lib/mailx/mailx.rc*
masterfile	Internet domain name server master data file	
mib.txt	Management information base for SNMP applications and SNMP agents	/etc/mib.txt
mnttab	Mounted file system table format	/etc/mnttab
motd	File that contains message of the day	/etc/motd*
named.boot	Domain name server configuration file	/etc/named.boot*
netgroup	List of network groups	/etc/netgroup
networks	Network name database	/etc/networks*
nl_types	Defines message system variables	nl_types.h
passwd	Password file format	/etc/passwd*
printcap	Printer capability database	/etc/printcap*
profile	Format of POSIX shell start-up file	/etc/profile*
proto	Prototype job file for at	/usr/lib/cron/.proto
protocols	Protocol name database	/etc/protocols*
publickey	Public key database	/etc/publickey*
queuedefs	Queue description file for at, batch, and cron	/usr/lib/cron/queuedefs

Man page	Description	File(s)
quota	Quota control file format	/sys/quota.h*
resolv.conf	Domain name resolver configuration file	/etc/resolv.conf*
rmtab	List of remotely mounted file systems	/etc/rmtab
sectab	Format for table of defined security names and values	/usr/include/sys/sectab.h
sendmail.cf	Configuration file for TCP/IP mail service	/usr/lib/sendmail.cf*
services	Network service name database	/etc/services*
share	Fair-share scheduler parameter table	/usr/include/sys/share.h
shells	List of available user shells	/etc/shells
text_tapeconfig	Tape subsystem configuration file	/etc/config/text_tapeconfig*
tapereq	User tape daemon interface	/usr/include/tapereq.h
tar	Tape archive file format	
term	Format of compiled term file	/usr/include/term.h
terminfo	Terminal capability database	/usr/lib/terminfo/*
tmpdir.users	List of authorized users for tmpdir(1)	/etc/tmpdir.users*
udb	Format of the user database file	/etc/udb /etc/udb.public
updaters	Configuration file for NIS updating	/etc/yp/updaters
utmp	utmp(5) and wtmp file formats	/etc/utmp /etc/wmtp
ypfiles	Network information service (NIS) database and directory structure	

Analyst Category

The man pages in this category describe files of interest to Cray Research analysts. Man pages in this subcategory describe internal files, including those that the UNICOS kernel uses as a reference. Cray Research or customer analysts do not change these files.

The files that an analyst sets up or modifies (to install or configure a UNICOS system) are not described in this manual; for more information on these files, see *General UNICOS System Administration*, Cray Research publication SG–2301.

Man page	Description	File(s)
acct	Per-process accounting file format	/usr/include/sys/acct.h
core	Core file format	
dir	Directory file format	/usr/include/sys/fs/c1dir.h

Man page	Description	File(s)
`dirent`	File-system-independent directory entry format	`/usr/include/sys/dirent.h`
`errfile`	Format of error-log file	`/usr/adm/errfile` `/usr/include/sys/err.h`
`fs`	File system partition format	`/usr/include/sys/fs/c1filsys.h`
`infoblk`	Loader information table	`/usr/include/infoblk.h`
`inode`	Inode format	`/usr/include/sys/ino.h`
`lnode`	Kernel user limits structure for fair-share scheduler	`/usr/include/sys/lnode.h`
`ipc`	Interprocess communication (IPC) access structures	`/usr/include/sys/ipc.h`
`msg`	Message queue structures	`/usr/include/sys/msg.h`
`sem`	Semaphore facility	`/usr/include/sys/sem.h`
`shm`	Shared memory facility	`/usr/include/sys/shm.h`
`slrec`	Security log record format	`/usr/include/sys/slrec.h`
`sysdump`	System dump files	`/core.sys`

NAME

acct – Per-process accounting file format

SYNOPSIS

```
#include <sys/types.h>
#include <sys/param.h>
#include <sys/accthdr.h>
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

Accounting files are produced if the acct(2) system call has enabled the system process accounting routine. The sys/acct.h include file gives the structure of these files.

Systems Accounting File Structure

The file structures differ slightly for accounting records on various Cray Research systems; the differences are indicated in the following file structures:

```
/*
 *      Kernel accounting structures.
 *
 *      Note: All of the structures in the unified accounting record
 *            must have the ac_flag field following the header.
 *
 */
typedef unsigned long comp_t;   /* 21-bit floating-point number   */
                                /* 5-bit exponent, 16-bit         */

/*
 *      Base-level accounting record.
 */
struct acctbs
{
        struct achead   ac_header;      /* header                         */
        unsigned        ac_flag:8;      /* accounting flags               */
        unsigned        ac_stat:8;      /* exit status                    */
        unsigned        ac_uid:24;      /* user ID                        */
        unsigned        ac_gid:24;      /* group ID                       */
        dev_t           ac_tty:32;      /* control typewriter             */
        time_t          ac_btime:32;    /* beginning time (seconds)       */
        comp_t          ac_utime:21;    /* user CPU time (clocks)         */
        comp_t          ac_stime:21;    /* system CPU time (clocks)       */
        comp_t          ac_etime:21;    /* elapsed time (clocks)          */
        comp_t          ac_mem:21;      /* 1st memory integral            */
                                        /* (click-tics)                   */
```

```
        comp_t          ac_mem2:21;     /* 2nd memory integral            */
                                        /* (click-tics)                   */
        comp_t          ac_mem3:21;     /* 3rd memory integral            */
                                        /* (click-tics)                   */
        comp_t          ac_io:21;       /* number of chars transferred    */
        comp_t          ac_rw:21;       /* number of physical I/O reqsts. */
        comp_t          ac_iowtime:21;  /* I/O wait time (clocks)         */
                                        /* runs while process is locked   */
                                        /* in memory                      */
        comp_t          ac_iowmem:21;   /* I/O wait time memory integral  */
                                        /* (click-tics) runs while        */
                                        /* process is locked in memory    */
        comp_t          ac_iosw:21;     /* I/O swap count                 */
        comp_t          ac_lio:21;      /* number of logical I/O requests */
        unsigned        ac_pid:21;      /* process ID                     */
        unsigned        ac_ppid:21;     /* parent process ID              */
        comp_t          ac_ctime:21;    /* process connect time (clocks)  */
        unsigned        ac_acid:24;     /* account ID                     */
        unsigned        ac_jobid:24;    /* job ID                         */
        unsigned        ac_nice:16;     /* nice value                     */
        char            ac_comm[8];     /* command name                   */
        comp_t          ac_iobtim:21;   /* I/O wait time (clocks)         */
        comp_t          ac_himem:21;    /* Hiwater memory mark (words)    */
        comp_t          ac_sctime:21;   /* system call time               */
};

/*
 *      End of Job record. Written when the last process is put to rest.
 */
struct  accteoj {
        struct achead   ac_header;              /* header                 */
        unsigned        ac_flag:8;              /* accounting flag        */
        unsigned        ace_jobid:24;           /* Job ID                 */
        unsigned        ace_uid:24;             /* User ID                */
        comp_t          ace_himem:21;           /* Hiwater mem. mark(clicks)*/
        comp_t          ace_sdshiwat:21;        /* SDS Hiwater mark       */
        unsigned        ace_nice:16;            /* Nice value             */
        long            ace_fsblkused;          /* #of fs blocks consumed   */
        time_t          ace_etime:32;           /* time at end of job     */
        comp_t          ace_shmint:21;          /* shmat integral (click-tics)*/
        comp_t          ace_shmsize:21;         /* shmget total size (words) */
};

/*
 *      Device specific I/O accounting record
 *      type field is filled in from superblock on block devices and
 *      from major number | ACCT_CHSP when used with a character device.
 */
```

```
#define NODEVACCT     8              /* devio entries per account*/
                                     /* records                  */
#define ACCT_CHSP     0200           /* marker for character     */
                                     /* special devices          */
#define ACCT_PERF     0400           /* marker for performance   */
                                     /* accounting               */
#define MAXPERFLVL    1              /* number of performance    */
                                     /* accounting levels        */

struct acctio {
       struct achead ac_header;      /* header                   */
       unsigned     ac_flag:8;       /* accounting flags         */
       struct {
              uint   acd_type:8;     /* major device no.         */
              comp_t acd_ioch:21;    /* characters transferred   */
              comp_t acd_lio:21;     /* logical I/O reqs count   */
       } ac_devio[NODEVACCT];
};
```

```
/*
 *     SDS accounting record (except on CRAY EL series)
 */
struct acctsds {
       struct achead  ac_header;       /* header                   */
       char           ac_flag;         /* accounting flag          */
       comp_t         acs_mem:21;      /* memory integral - based */
                                       /* on residency time,       */
                                       /* not execution time       */
       comp_t         acs_lio:21;      /* logical I/O reqs count  */
       comp_t         acs_ioch:21;     /* chars transferred        */
       comp_t         acs_memsw:21;    /* mem integral - suspend/resume */
};
```

The following MPP accounting record of the `acct` file is for use only with Cray MPP systems:

```
/*
 *      MPP accounting record.
 */
struct  acctmpp {
        struct achead   ac_header;      /* header                   */
        char            ac_flag;        /* accounting flag          */
        unsigned        ac_mpppe:16;    /* MPP processing elements */
        unsigned        ac_mppbb:8;     /* MPP barrier bits         */
        comp_t          ac_mpptime:21;  /* MPP time (in clocks)     */
};
```

The following multitasking accounting record substructure is shared by all Cray Research systems, and the record structures for specified systems are given.

```
/*
 *      Multitasking accounting record substructure.
 */
struct  mu {
        uint              :1;
        comp_t          m0:16;
        comp_t          m1:16;
        comp_t          m2:16;
        comp_t          m3:16;
};

struct  acctmu {
        struct achead   ac_header;          /* header                        */
        unsigned        ac_flag:8;          /* accounting flag               */
        long            ac_smwtime;         /* semaphore wait time (clocks) */
        struct mu       ac_mutime[MUSIZE];  /* time (compressed) connected  */
                                            /* to (i+1) CPUs (1/100 sec)    */
};
```

The rest of the acct file applies to all Cray Research systems, except as specified:

```
/*
 *     Error accounting record.
 */
struct accter {
       struct achead  ac_header       /* header                          */
       unsigned       ac_flag:8;      /* accounting flag                 */
       short          ac_errno;       /* u_error returned from writei()  */
       struct acerror ac_error;       /* error info from writei()        */
};

/*
 *     Performance accounting record.
 */
struct acctperf {
     struct achead  ac_header;         /* header                            */
     unsigned       ac_flag:8;         /* accounting flag                   */
     comp_t         acp_rtime:21;      /* process start time (in clocks)  */
                                       /* past ac_btime                     */
     comp_t         acp_tiowtime:21;   /* terminal I/O wait time            */
     comp_t         acp_srunwtime:21;  /* SRUN wait time (in seconds)       */
     comp_t         acp_swapclocks:21; /* swapped time (in clocks)          */
     long           acp_rwblks:21;     /* # of bufrd physical blks moved  */
     long           acp_phrwblks:21;   /* # of raw physical blks moved     */
};

/*
 *     Unified accounting record.
 */
union acct {
       struct acctbs    acctbs;
       struct acctio    acctio;
       struct acctmu    acctmu;
       struct accter    accter;
       struct acctsds   acctsds;
       struct acctmpp   acctmpp;
       struct acctperf  acctperf;
       struct accteoj   accteoj;
};

#ifdef  KERNEL
extern struct acctind acctp[];          /* inode of accting files   */
#endif                                  /* KERNEL                   */

/*
 *      Maximum number of acct records per process.
 *
```

```
 *      1 Base record + 1 Multitasking record + 1 SDS record + 1 MPP record +
 *      1 performance record + _MAXDEVIOREC device.
 *      Note the end of job record is not added since it is always singular.
 */
#define _MAXDEVIOREC ((MAXBDEVNO + MAXCDEVNO + NODEVACCT - 1)/NODEVACCT)
#define NOACCTREC       (1+1+1+1+1+_MAXDEVIOREC)

/*
 *      Flag definitions, for ac_flag.
 */
#define AFORK          01                /* has executed fork         */
                                         /* but no exec               */
#define ASU            02                /* used super-user privileges*/
#define AMORE          04                /* more accounting records   */
                                         /* follow for this process   */
#define ACCTR          0370              /* record type               */
#define ACCTBASE       0000              /* base-level acctg records  */
#define ACCTIO         0010              /* device-specific I/O       */
                                         /* accounting record         */
#define ACCTMU         0020              /* multitasking acctg record */
#define ACCTERR        0030              /* error accounting record   */
#define ACCTSDS        0040              /* SDS accounting record     */
#define ACCTEOJ        0050              /* EOJ accounting record     */
#define ACCTPERF       0060              /* performance accting rcrd  */
#define ACCTMPP        0070              /* MPP accounting record     */

/*
 *      Function types for devacct system call.
 */
#define ACCT_ON        1                 /* Device accounting on      */
#define ACCT_OFF       2                 /* Device accounting off     */
#define ACCT_LABEL     3                 /* Label block special       */
                                         /* device                    */
#define PERF_01        1                 /* Additional performance    */
                                         /* accounting on             */

#ifdef  KERNEL
/*
 *     Accounting file vnode pointers.
 */
struct acctind {
       int            did;               /* daemon identifier         */
       struct vnode   *vno;              /* acct file vnode pointer   */
};
#endif                                   /* KERNEL                    */
#include <sys/cdefs.h>

__BEGIN_DECLS
```

```
extern int devacct __((char *_Device, int _Func, int _Type));
__END_DECLS

#endif                                  /* KERNEL                   */
```

FILES

/usr/include/sys/acct.h	Structure of per-process accounting files
/usr/src/cmd/acct/include/cacct.h	Structure of condensed accounting files
/usr/src/cmd/acct/include/session.h	Structure of session record files
/usr/src/cmd/acct/include/tacct.h	Structure of per-process total accounting files

SEE ALSO

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

acct(2), devacct(2), exec(2), fork(2) in the *UNICOS System Calls Reference Manual*, Cray Research publication SR–2012

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

NAME

acid – Format of the account ID information file

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The /etc/acid file contains the following information for each account:

- Account name
- Account ID

The acid file is an ASCII file, which resides in the /etc directory. The fields are separated by colons; each account record is separated from the next by a newline character. udbgen(8) maintains the acid file automatically to match the information in the udb file.

The acid file maps numeric account IDs (called *ACID*s in the UDB) to account names. The account names belong to the accounting subsystem and are not user names.

NOTES

Unlike the /etc/passwd file, you must update the /etc/acid file manually to include new account IDs and account names. When you update /etc/acid, ensure that the udbgen(8) utility is not running, because udbgen would overwrite any changes to /etc/acid.

FILES

/etc/acid	Format of account ID information file

SEE ALSO

acct(5), group(5), passwd(5), udb(5)

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

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

NAME

`acl` – User access control lists format

SYNOPSIS

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

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

An access control list (ACL) is a mechanism for user (discretionary) file access control. An ACL contains entries that define the allowed access to a file on a specific user and/or group basis.

To create and maintain an ACL file, use the `spacl`(1) command. You can use the `spset -a` command to assign an existing ACL file to a file or list of files. The `spclr -a` command removes an ACL from a file or list of files.

An ACL file consists of multiple entries, one entry per user/group name pair. Each entry has the following format:

user`:`*group*`:`*permissions*

user User name; to represent all users, use an `*` symbol.

group Group name; to represent all groups, use an `*` symbol; to represent the owning group, use ( ).

permissions Permissions for access. Permissions are specified as follows:

- `r` Grants read permission
- `w` Grants write permission
- `x` Grants execute permission
- `n` Denies access

You can specify any combination of `r`, `w`, and `x`, or `n`.

You can specify `()` for the group only when the specified user is `*`. You cannot specify `*` for both user and group.

The format of an ACL file is defined in the `sys/acl.h` include file, as follows:

```
struct  acl  {
        uint    ac_usid :24,    /* user ID */
                ac_grid :24,    /* group ID */
                ac_flag :4,     /* ACL entry type */
                ac_mode :4;     /* access mode - r/w/x */
                ac_sort :2,     /* sort flag */
                ac_same :6;     /* same uid count */
};

struct  acl_rec  {
        long    ac_magic;
        uint    ac_size   :24,
                ac_owner  :24,
                ac_type   :8,
                ac_fill   :8;
        uint    ac_links  :24,
                ac_gmode  :3,
                ac_vsn    :6,
                ac_resrv  :31;
        struct acl acl[ACLSIZE];
};

#define ACLMAGIC     0xac0ff12ee21ff0ca
/*
 *      ACL entry types
*/
#define FLAG_UIDGID  01         /* uid.gid acl entry */
#define FLAG_GIDONLY 02         /* gid only acl entry */
#define FLAG_UIDONLY 04         /* uid only acl entry */
#define FLAG_OWNGID  010        /* owning group ACL entry */
```

NOTES

The file's group permission bits are used as a mask, which is intersected with the ACL entry permissions to determine the allowed access. This means that the group permission bits of the file always show the maximum amount of access allowed any user and/or group specified in the ACL. You must specify the permissions in both the mask and ACL entry to be allowed. For example, if the file's permission bits are set to 750 (that is, the group bits are set to r-x) and a user's ACL entry is set for read and write access only (rw-), the user is allowed only read access to that file. The user is not allowed write or execute access because both entries did not specify these permissions.

For a complete description of the masking operation and the order that ACL entries are checked, see the Security section in the *General UNICOS System Administration*, Cray Research publication SG–2301.

For a description of ACL creation and maintenance operations, see the `spacl`(1) command.

EXAMPLES

Example 1: The following ACL entry defines read, write, and execute permission to user `fred`, in any group:

```
fred:*:rwx
```

Example 2: The following ACL entry defines user `betty` read and write permission when she is in group `admin`:

```
betty:admin:rw
```

Example 3: The following ACL entry denies user `ralph` any permissions, in any group:

```
ralph:*:n
```

Example 4: The following ACL entry defines read access for owning group:

```
*::r:
```

The ACL mask is intersected with the ACL entry to determine the type of access granted.

FILES

`/usr/include/sys/acl.h` Format of user access control lists

SEE ALSO

`slog`(4), `slrec`(5)

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

General UNICOS System Administration, Cray Research publication SG–2301

NAME

aft – ASCII flaw table

IMPLEMENTATION

Cray PVP systems with an IOS model E

DESCRIPTION

The files in /etc/aft contain information about physical disk defects. One aft file represents each physical device. The aft files are used by the bb command, which translates physical disk addresses into logical relative block addresses.

The files in /etc/aft are named for the I/O paths of the physical devices they represent. They are created by the ift(8) command.

The aft files may be edited to add, delete, or change entries. They can then be used to initialize the physical device spare sector maps by using the spmap(8) command.

EXAMPLES

A typical aft file follows:

```
*
*  engineering flaw table for DD-49
*
*  factory flaw map date:  09-08-89
*
*       S/N     7009

#       0 0 0 0

*       count   head    sector  cylinder

        1       7       43      1307
        1       1       47      1547
        1       1       50      1547
        1       1       50      1557
```

To initialize an aft file, enter the following command line:

```
ift /dev/ift/0130.1 >/etc/ift/0130.1
```

FILES

`/etc/aft/*`

SEE ALSO

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

NAME

aliases – Defines alias database for sendmail(8)

SYNOPSIS

/usr/lib/aliases

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The /usr/lib/aliases file defines the alias database for sendmail(8). The format of this file consists of the following:

alias_name: *recipient_1*, *recipient_2*, *recipient_3*, . . .

alias_name is the name to alias, and *recipient_n* is the alias for that name. Lines beginning with white space (spaces or tabs) are continuation lines.

Aliasing occurs only on local names. Loops can not occur, since no message will be sent to any person more than once.

After aliasing has been done, local and valid recipients who have a .forward file in their home directory will have their messages forwarded to the list of users defined in that file.

This is only the raw data file; the actual aliasing information is placed into a binary format in the /usr/lib/aliases.pag file, using the program newaliases(1). A newaliases command must be executed each time the aliases file has been changed before the changes will take effect.

NOTES

Blank lines and lines that begin with a # are comments.

The file must contain an alias for Postmaster and MAILER_DAEMON.

EXAMPLES

The following is an example of entries in an /usr/lib/aliases file:

```
Postmaster: root
MAILER-DAEMON: postmaster
```

FILES

`/usr/lib/aliases` File that contains the alias database for `sendmail`(8)

SEE ALSO

`newaliases`(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011
`dbm`(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080
`sendmail`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

a.out – Loader output file

SYNOPSIS

`#include <a.out.h>`

IMPLEMENTATION

All Cray Research systems except Cray MPP systems.

DESCRIPTION

The a.out file is the output file generated when the ld(1) or segldr(1) loader command is executed. If all errors occurring during the load process were at the caution level or lower, both commands make file a.out executable.

When the UNICOS operating system executes an a.out file that does not use shared text, it loads the file, as follows:

1. Reads a_text words into common memory at location a_origin (usually 0); see the header format that follows.
2. Reads the following a_data words of initialized data in at location a_origin + a_text.
3. Fills a_bss words at location a_origin + a_text + a_data with 0's.
4. Begins execution at parcel address a_entry.

When the UNICOS operating system executes an a.out file that uses shared text, it loads the file, as follows:

1. Reads a_text words into the instruction address space at location a_origin (usually 0); see the header format that follows.
2. Reads a_data words of initialized data into memory at address 0 of the data address space.
3. Fills a_bss words at location a_origin + a_data in the data space with 0's.
4. Begins execution at parcel address a_entry in the instruction space.

The following shows the header format:

```
struct exec      {
   union  {
     long   omagic;        /* old magic number                      */
     struct {
       unsigned     :32;   /* new, reserved - must be zero          */
       unsigned st  : 1;   /* new, shared text indicator            */
       unsigned     : 7;   /* new, reserved - must be zero          */
       unsigned pmt : 8;   /* new, primary machine type             */
       unsigned id  :16;   /* magic identifier                      */
     } nmagic;
   } u_mag;
   long    a_text;         /* size of text area in words            */
   long    a_data;         /* size of data area in words            */
   long    a_bss;          /* size of bss area in words             */
   long    a_syms;         /* size of symbol table in words         */
   long    a_entry;        /* entry point (parcel address)          */
   long    a_origin;       /* old base address (usually zero)       */
   union {
     long  ofill1;         /* flag, 1 = relocation info stripped */
     struct {
       unsigned ptr  :32;  /* new, byte offset of _infoblk          */
       unsigned      :31;  /* new, reserved - must be zero          */
       unsigned str  : 1;  /* new, stripped bit                     */
     } info;
   } u_fill1;
};

/* defines for compatibility */
#define   a_magic         u_mag.nmagic.id
#define   a_omagic        u_mag.omagic
#define   a_fill1         u_fill1.info.str

/* defines for new fields */
#define   a_id            u_mag.nmagic.id
#define   a_st            u_mag.nmagic.st
#define   a_pmt           u_mag.nmagic.pmt
#define   a_infoptr       u_fill1.info.ptr
#define   a_str           u_fill1.info.str

#define   A_MAGIC1        0407  /*  normal magic                   */
#define   A_MAGIC2        0410  /*  shared text                    */
#define   A_MAGIC3        0411  /*  normal ymp-32 bit magic        */
```

```
#define   A_MAGIC4        0412  /*  shared text ymp-32 bit magic    */

#define   A_MAGIC_ID      0407  /* new magic id                     */
/* --- primary machine types ---*/
#define   A_PMT_UNDF      0     /* undefined machine type =>old hdr */
#define   A_PMT_INC       1     /* incremental load code fragment   */
#define   A_PMT_CRAY1     2     /* CRAY-1S                          */
#define   A_PMT_XMP_NOEMA 3     /* CRAY-X/MP, 22-bit mode           */
#define   A_PMT_XMP_ANY   4     /* CRAY-X/MP, mode indifferent      */
#define   A_PMT_XMP_EMA   5     /* CRAY-X/MP, 24-bit mode           */
#define   A_PMT_CRAY2     6     /* CRAY-2                           */
#define   A_PMT_YMP       7     /* CRAY-Y/MP                        */
#define   A_PMT_C90       8     /* CRAY C90                         */
```

NOTES

The UNICOS object file format is unique. AT&T common object files are not supported.

FILES

`/usr/include/a.out.h`	Default, executable, output file header format, which the `ld`(1) and `segldr`(1) commands produce

SEE ALSO

`mpp.a.out`(5) for the description of the Cray MPP loader `a.out` file
`relo`(5) for information about the relocatable object table format under the UNICOS operating system

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

Segment Loader (SEGLDR) and ld Reference Manual, Cray Research publication SR–0066

NAME

ar – Archive file format

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

This entry describes the format of an archive file. The ar(1) archive command combines several files into one. You can use archives as libraries through which the link editors ld(1) and segldr(1) search; however, the bld(1) utility is recommended for this purpose.

Each archive begins with the following archive magic strings:

```
#define  ARMAG  "!<arch>\n"   /* magic string                       */
#define  SARMAG 8             /* length of magic string             */
```

The individual files, which are called *archive file members*, follow the archive magic string. Each file member is preceded by a file member header, which has the following format:

```
#define  ARFMAG  "`\n"        /* header trailer string              */

struct ar_hdr                 /* file member header                 */
{
         char  ar_name[16];   /* '/' terminated file member name  */
         char  ar_date[12];   /* file member date                   */
         char  ar_uid[6];     /* file member user identification  */
         char  ar_gid[6];     /* file member group identification */
         char  ar_mode[8];    /* file member mode (octal)           */
         char  ar_size[10];   /* file member size                   */
         char  ar_fmag[2];    /* header trailer string              */
};
```

All information in the file member headers is in printable ASCII. The numeric information contained in the headers is stored as decimal numbers (except for ar_mode, which is in octal). Thus, if the archive contains printable files, you can print the archive itself.

The ar_name field is blank-filled and slash (/) terminated. The ar_date field contains the modification date of the file at the time of its insertion into the archive. If you use the ar(1) portable archive command, you can move common format archives from system to system.

Only the name field has any provision for overflow. If any file name consists of more than 14 characters or contains an embedded <space>, the string "#1/" followed by the ASCII length of the name is written in the name field. The file size (stored in the archive header) is incremented by the length of the name. The name is then written immediately following the archive header.

Each archive file member begins on an even-byte boundary; if necessary, `<newline>` characters are inserted between files. If the file name is less than or equal to 14 characters and does not contain an embedded `<space>`, the size specified (`ar_size`) reflects the actual size of the file, exclusive of padding. Otherwise, the size specified reflects the actual size of the file, plus the number of characters in the file name.

No provision exists for empty areas in an archive file.

FILES

`/usr/include/ar.h` Format of archive files

SEE ALSO

`a.out`(5) for loader output file information

`ar`(1) which is the archive and library maintainer for portable archives
`bld`(1) to maintain relocatable libraries
`segldr`(1) to invoke the Cray Research segment loader (SEGLDR)
in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

NAME

`arrayd.conf`, `arrayd.auth` – Array services configuration files

SYNOPSIS

`/usr/lib/array/arrayd.conf`

`/usr/lib/array/arrayd.auth`

IMPLEMENTATION

IRIX and UNICOS systems

DESCRIPTION

The `arrayd.conf` and `arrayd.auth` files describe the configuration of one or more arrays. The default configuration files are `/usr/lib/array/arrayd.conf` and `/usr/lib/array/arrayd.auth`, although the system administrator can override this or specify additional files. Every machine running an array services daemon (which should be every machine that is part of an array) must have its own configuration file or files. The configuration files contain information about which arrays are known to the array services daemon and the machines in each of them, the commands that can be executed by the array services daemon, various local options, and information used for authenticating messages passed between array services daemons on different machines.

The `arrayd.conf` file is typically readable by all users, while the `arrayd.auth` file is generally readable only by root. Other than their initial access permissions upon installation, there is no functional difference between the two files; either may contain any sort of configuration information. However, because `arrayd.auth` is not readable by most users, it is most appropriate for secure information such as authentication keys, while `arrayd.conf` is intended to contain public information such as the array and command definitions.

The initial configuration files that are installed with array services are very minimal. These files describe a single array, made up only of the local machine, and no authentication. Every site installing array services will need to customize the configuration file to describe its local arrangement.

General Syntax

The configuration file itself is made up of regular, human-readable ASCII text. Blank lines and comments (introduced by a # character) are ignored. There are four types of entries in the configuration file: array definitions, command definitions, local options, and authentication information. A typical entry may consist of several subentries; by convention, each should be on a separate line. Similarly, some subentries may have options, which should be on separate lines as well. Leading white space is ignored, so subentries and options can (and should) be indented for improved readability. The entries in a configuration file and the subentries within an individual entry need not be in any particular order.

Arguments

Most of the various entries, subentries, and options take arguments. `arrayd.conf` accepts the following arguments:

names These are simple identifiers, similar to variable names. They can contain upper- and lower-case letters, the characters `-` and `_`, and numeric digits (although the first character must not be a digit).

numbers These are treated as signed 64-bit integers and may be specified in hexadecimal, octal, or decimal, with hexadecimal values being preceded by `0x` and octal values being preceded by `0`.

environment variables
A name preceded by a `$` is presumed to refer to an environment variable and will be substituted accordingly.

strings Any arbitrary string of characters enclosed in double quotes. Double quotes and backslashes can be embedded within the string by preceding them with a backslash. Newlines and tabs can be included using `"\n"` and `"\t"`, respectively. A real newline may also be embedded by preceding it with a backslash, thus allowing a string to span several lines in a configuration file.

substitution variables
A name preceded by a `%` is referred to as a *substitution variable* and will be replaced with some other value. Recognized substitution variables include the following:

`%1`, `%2`, ..., `%9`
These represent the first nine arguments specified for an `array` command. For example, if a user invokes an `array` command with `array killjob 1354 token`, then `%1` would be replaced with `1354` (the first argument to the command `killjob`), and `%2` would be replaced with `token`. Arguments that do not exist (`%3` in this case) are replaced with an empty string.

`%ALLARGS`
This is replaced with all of the arguments that were specified for an `array` command. When used with subentries that take multiple arguments, each individual command-line argument is treated as an individual argument in the subentry as well. When used with subentries that take only a single argument, only the first command-line argument is actually substituted.

`%ARRAY`
This is replaced with the name of the array that is the target of the current `array` command. This is primarily of use when a machine belongs to two or more separate arrays.

`%ASH` This is replaced with the array session handle of the program that invoked the current `array` command. It is in hexadecimal and is preceded by the string `0x`.

`%GROUP`
This is replaced with the name corresponding to the effective group ID of the process that invoked the current `array` command.

`%LOCAL`
This is replaced with the name of the local machine, as specified in a `LOCAL HOSTNAME` entry. This is useful if several machines share a configuration file containing commands.

`%ORIGIN`
This is replaced with the primary hostname of the network interface that transmitted the request from the client machine. If the client and server are the same machine, then this is `localhost`. This is often not the same as the client's machine name, as it typically includes the network name as well (for example, `machine.domain.com`, not just `machine`).

`%OUTFILE`
This variable is valid only as part of a merge command. It is replaced with a list of one or more temporary files. Each file contains the output from a single machine of the related `array` command. When the merge command is finished, the temporary files are automatically removed. The files in the list are not in any particular order; if the merge command needs to know which machine a specific file came from, the original `array` command should include that data in its output. When used with subentries that take multiple arguments, each individual pathname is treated as an individual argument in the subentry as well. When used with subentries that take only a single argument, only the first output file pathname is actually substituted.

`%REALGROUP`
If the process that invoked the current `array` command has different real and effective group IDs, then this is replaced with the name corresponding to the real group ID. If the real and effective group IDs are the same, then `<same>` is substituted instead.

`%REALUSER`
If the process that invoked the current `array` command has different real and effective user IDs, then this is replaced with the name corresponding to the real user ID. If the real and effective user IDs are the same, then `<same>` is substituted instead.

`%USER` This is replaced with the name corresponding to the effective user ID of the process that invoked the current `array` command.

Note that the names of these substitution variables may be in either upper- or lower-case. If an unrecognized variable name is specified, a warning is issued, and the variable is replaced with an empty string.

substitution functions
A substitution variable followed immediately by one or more arguments enclosed in parentheses is a *substitution function*. An argument to a substitution function can generally be anything that is valid as the argument to an entry or subentry, except for another substitution function. Recognized substitution functions include

`%ARG`(*number*)
: This is replaced with the command argument specified by *number*, which should be a numeric value. If the argument does not exist, a warning is generated, and an empty string is substituted.

`%OPTARG`(*number*)
: This is similar to `%ARG`(...), except that no warning is generated if the specified argument does not exist. This is useful for specifying optional arguments.

`%PID`(*ash*)
: *ash* specifies an array session handle. This is replaced with a list of all process IDs (PIDs) that belong to the specified array session on the local machine. For entries that take more than one argument, each PID is treated as a separate argument (see `%ALLARGS`).

As with *substitution variables*, an unrecognized substitution function is replaced with an empty string and causes a warning to be generated.

literal arguments
: A *literal argument* is any argument that can be evaluated when the array services daemon is first started. This includes names, strings, numbers, and environment variables, but specifically does not include substitution variables or functions.

numeric arguments
: A *numeric argument* is an argument that can be resolved to a numeric value when the array services daemon is first started. This includes actual numbers, as well as strings and environment variables. An error occurs if a string or environment variable cannot be converted to proper numeric values.

Array Entries

An array entry is a configuration file entry that defines the machines and other details that make up a particular array. The general format is as follows:

`ARRAY` *array-name*
`ARRAY_ATTRIBUTE` *name=value*
`ARRAY_ATTRIBUTE` *litarg*. . .
`IDENT` *number*
`SEQFILE` *pathname*
`MACHINE` *machine-name-1*
machine options
. . .
`MACHINE` *machine-name-2*
. . .

Keywords such as `ARRAY`, `MACHINE`, and `IDENT` may be in either upper- or lower-case; upper-case is used here to distinguish them from other fields. The various subentries do not necessarily have to occur in any particular order. However, they should not appear between options in a `MACHINE` subentry.

array-name is the name that will be used to refer to the array as a whole; it may be of any length. This is the name that would be used with the `-a` option of the `array`(1) command.

The `ARRAY_ATTRIBUTE` subentry is used to specify one or more arbitrary values that will be maintained in the configuration database, but will otherwise be ignored by the array services daemon. Programs that obtain array configuration information (for example, using the `aslistarrays`(3X) function) will be provided with a list of these attributes. Thus, these could be useful for maintaining miscellaneous configuration information that may be needed by other programs. The `ARRAY_ATTRIBUTE` subentry may be specified more than once. If the attribute starts with a simple identifier followed by an equal sign, then the remainder of the line (with multiple blanks and tabs converted to a single space) is appended to form a single attribute. Such an attribute could be used along with the `asgetattr`(3X) function in a manner similar to environment variables. If the attribute is formed of any other literal argument, it is presumed to end as soon as white space is encountered. In this case, multiple attributes could be specified on a single line.

The `SEQFILE` subentry specifies the pathname of a file used to keep an array session sequence number for the array. The default sequence file is located in the directory specified by `LOCAL DIR` (see below) and has a name formed by appending the array name to the string `.seqfile.`.

The `IDENT` subentry specifies a numeric value that is used when generating global array session handles for the array. No other array should have the same `IDENT` value. If an `IDENT` value is not specified, a random one will be generated. The value should be in the range of 1 to 32767.

Each `MACHINE` subentry specifies a single machine that is a member of the array. Each `ARRAY` entry must have at least one `MACHINE` subentry. *machine-name* is the name that is used to refer to this machine. Ordinarily this would be the machine's host name; however, that is merely a convention and not a requirement. A `MACHINE` subentry may have zero or more options. These include

`MACHINE_ATTRIBUTE` *litarg*. . . or *name*=*value*

The `MACHINE_ATTRIBUTE` option is similar to the `ARRAY_ATTRIBUTE` subentry in that it is used to specify one or more arbitrary values that are maintained in the configuration database, but otherwise are ignored by the array services daemon. Programs that obtain machine configuration information (for example, using the `aslistmachines`(3X) function) are provided with a list of these attributes. Thus, these are useful for maintaining miscellaneous configuration information that may be needed by other programs. The `MACHINE_ATTRIBUTE` option may be specified more than once, and it has the same syntax as `ARRAY_ATTRIBUTE`.

`[SERVER] HOSTNAME` "*string*"

This specifies the full host name or IP address of the machine. The value should be enclosed in double quotation marks. If a `HOSTNAME` is not specified, the machine name will be used. The string `SERVER` is optional.

`SERVER IDENT` *number*

This specifies the numeric identifier of the array services daemon on the specified machine. This value may be used for generating global array session handles or uniquely identifying the machine. If a `SERVER IDENT` is specified for a machine, it should match the `LOCAL IDENT` that is specified in that machine's local array services configuration file. Unlike the syntax for the `HOSTNAME` and `PORT` options, the string `SERVER` that comes before `IDENT` is required.

`[SERVER] PORT` *number*

This specifies the port on which the array services daemon for this machine is listening. This would override the default port number of 5434. The string `SERVER` is optional.

Command Definitions

A command entry defines the actual program that is invoked by the array services daemon when it receives an `array` command. Its format is similar to that for an array entry:

COMMAND *cmd-name*
INVOKE *any-args*. . .
MERGE *any-args*. . .
GROUP *any-arg*
USER *any-arg*
OPTIONS *litarg*. . .

cmd-name specifies the actual command name. This is what the user would use when invoking the command with `array`(1).

The `INVOKE` subentry specifies the actual program to be executed, plus any arguments that should be supplied to it. Any number of arguments may be specified for the `INVOKE` subentry. Groups of arguments that are not separated by white space are concatenated to form single values (white space embedded in a string is not considered to be white space for these purposes). Each resulting value is passed to the program to be executed as a single argument. Thus, if a user typed `array foo a b c`, and the `INVOKE` subentry for the command `foo` were as follows:

```
INVOKE /usr/bin/test%1 %2"this is a test" %3
```

The argument list for the program to be executed would consist of the following:

```
argv[0] = "/usr/bin/testa"
argv[1] = "bthis is a test"
argv[2] = "c"
```

The first value in the argument list also specifies the actual pathname of the program to be executed (`/usr/bin/testa` in this case). The array services daemon does not have a search path, so this must specify either an absolute path to the file to be executed, or a path relative to the array services daemon's current directory (see the `DIR` local option).

The MERGE subentry is used to specify a merge command. Ordinarily, when an array command is run on several machines, the results and output from each machine are returned as separate streams of data. However, if a merge command is specified, it is run after the array command itself has been completed on all machines, and only the results and output of the merge command are returned. When used with the %OUTFILES substitution variable, this could be a convenient way to consolidate or summarize the results of the array command. The MERGE command is executed in the same way as a normal INVOKE command, except that it always runs on the same machine as the array services daemon, even if that particular machine is not a member of the array on which the array command was run.

The GROUP and USER subentries are optional and specify the name of the group and user under which the program should be run. Each of these take a single argument. To run with the IDs of the user who invoked the array command, these could be specified as %GROUP and %USER, respectively. If these are not specified for a particular command entry, they default first to the values set in the local options, or, if those are not present, to user and group guest. By default, the GROUP and USER subentries affect only the effective group and user IDs of the program; the real group and user IDs will be the same as those of the process that invoked the program. This behavior can be changed by using the SETRGID and SETRUID command options (see below).

The OPTIONS subentry is used to specify additional details about how the command should be processed. It should be followed by one or more arguments from the following list. The arguments may be in either upper- or lower-case. They may also be preceded by the string NO to negate their effects.

LOCAL
: Executes the command on the same machine as the array services daemon only, even if a target array was specified explicitly or by default.

NEWSESSION
: Executes the command in a new global array session. Normally the command would be run in the same array session as the process that invoked it.

QUIET
: Discards any output generated by the command. If a merge command has been specified, QUIET applies to the merge command and not the invoke command. This would allow a merge command to quietly act on the output of the invoke commands.

SETRGID
: Runs the command with both its real and effective group IDs set to the value specified by the GROUP subentry. Normally, only the effective group ID is taken from the GROUP subentry, while the real group ID is taken from the process that invoked the command.

SETRUID
: Runs the command with both its real and effective user IDs set to the value specified by the USER subentry. Normally, only the effective user ID is taken from the USER subentry, while the real user ID is taken from the process that invoked the command.

WAIT
: Waits for each invoked program to complete execution before returning control to the process that requested the command. This is the default behavior. If NOWAIT is specified, control is returned to the requester immediately after the invoked programs are started. NOWAIT implies QUIET and causes any merge command to be ignored.

Local Options

A local options entry specifies options to be used by the array services daemon itself. If more than one local options entry is specified, settings in later entries silently override those in earlier entries. A local options entry looks like this:

`LOCAL`

> `DIR` *literal-arg* `DESTINATION ARRAY` *literal-arg* `GROUP` *literal-arg*
> `HOSTNAME` *literal-args* `IDENT` *num-arg* `PORT` *num-arg*
> `USER` *literal-arg* `OPTIONS` *literal-arg*`...`

All of the subentries in a local entry are optional.

`DIR` Specifies an absolute pathname for the array services daemon's working directory. The default is `/usr/lib/array`.

`DESTINATION ARRAY`
: Specifies the default target array for `array` commands when one has not been specified explicitly by the user. There is no default value unless only one array is defined (in which case it becomes the default); if a user omits the target array and there is no default, an error occurs.

`GROUP` and `USER`
: Specify the names of the group and user under which an `array` command should be run. A `GROUP` or `USER` specified in a particular command entry always overrides these values. These subentries default to the group and user that is running the array services daemon.

`HOSTNAME`
: Specifies the value that is returned by the `%LOCAL` substitution variable. The results of array services commands initiated with `ascommand`(3X) also refer to this name. The default is the actual host name of the local machine.

`IDENT` Specifies a numeric value that is included in global array session handles generated by this array services daemon. Some versions of UNICOS may also make use of this value to generate their own global array session handles. No other array services daemon should have the same `IDENT` value. If an `IDENT` value is not specified, one is generated from the `hostid` of the local machine. The value must be in the range of 1 to 32767.

`PORT` Specifies the network port on which this array services daemon listens for requests. The default is the standard `sgi-arrayd service`, 5434.

`OPTIONS`
: Specifies additional details about the operation of the array services daemon. It should be followed by one or more arguments from the following list. The arguments may be in either upper- or lower-case. They may also be preceded by the string `NO` to negate their effects.

`CHKLOCALID` Instructs `arrayd` to make certain authentication checks when accepting a connection from a local user, such as ensuring that the user is formally authorized for their current group. Note that these checks may fail on systems that have mechanisms for changing the real group of a user to a setting that is not in one of the standard administrative files (for example, `/etc/group` or its corresponding network information service (NIS) map).

`SETMACHID` Some versions of UNICOS permit setting a system machine identifier, which is used by the kernel for generating global array session handles. If the current system has this facility, and `SETMACHID` is specified, `arrayd` sets the machine ID to the value specified by a `LOCAL IDENT` statement in the configuration file or on the command line with the `-m` option.

`SVR4SIGS` Instructs `arrayd` to use SVR4 semantics for the `SIGXCPU` and `SIGXFSZ` signals when starting a new process to handle a remote execution request (such as those issued by the Array Services `arshell`(1) command). In this mode, the new process ignores `SIGXCPU` and `SIGXFSZ` signals unless it specifically alters the behavior for those signals with a system call such as `signal`(2) or `sigset`(2). This is different from the default behavior for processes started by `arrayd` to handle remote execution requests, in which `SIGXCPU` and `SIGXFSZ` will cause the process to abort with a core dump. (This feature requires the Array Services 3.1 for IRIX release or later.)

Authentication Information

An authentication information entry is used to describe the type of authentication that should be done when passing messages to and from another array services daemon. Authentication information entries do not accumulate: if more than one is encountered in the various configuration files processed by an array services daemon, only the last one has any effect; all information from previous entries is discarded. There is currently only one type of authentication provided, although more may be provided in the future. Its entry is as follows:

`AUTHENTICATION SIMPLE`

`HOSTNAME` *literal-arg* `KEY` *num-arg* `HOSTNAME` *literal-arg* `KEY` *num-arg* . . .

This entry contains one or more subentries consisting of machine/key pairs. *literal-arg* is the network host name of a machine. Notice that the network host name is not necessarily the same as the machine name used to identify a machine in an array entry (see above). *num-arg* is a 64-bit unsigned integer that is to be used as the authentication key for all messages originating from that machine. If a key of `0` is specified, authentication is not performed on messages originating from that machine. Similarly, if a machine has no subentry at all, no authentication is performed on messages received from it.

If a machine appears in more than one array entry, it needs to have only one subentry in the authentication information. Conversely, the machine in an authentication information subentry does not need to appear in any array entries.

With the `SIMPLE` scheme, a *digital signature* is calculated for each message by using the authentication key associated with the sending machine, and this value is then sent along with the message. When an array services daemon receives a message from another machine, it checks its private database for the authentication key associated with the machine that sent the message, recalculates the digital signature, and ensures that it matches the one sent with the message. This provides some basic protection against forged messages because a forger (presumably) would not have access to the authentication key that is required to calculate a proper digital signature.

Because this approach depends on the secrecy of the authentication keys, it is important to put this type of authentication information entry in a configuration file that is not accessible to general users (for example, the `arrayd.auth` file in the default installation). Because both the sender and receiver need to have the same authentication key for a given machine, the administrator must take special care to ensure that the authentication information in each machine's configuration files is consistent with that in the corresponding file.

There are some circumstances in which array services may be needed on an array of only one machine (for example, systems that use the MPI message passing library). For these systems, an alternative to using simple authentication is to simply disallow any requests from remote systems. This can be done by specifying an authentication information entry of the form

```
AUTHENTICATION NOREMOTE
```

For the purposes of array services, any request to an IP address other than 127.0.0.1 is considered to be remote. Therefore, the `HOSTNAME` entry for the local machine in any array should be either `127.0.0.1` or `localhost` if `NOREMOTE` is being used. While this blocks any incoming array services requests from remote machines, it does not prevent outgoing array services requests originating on the local machine from being sent to remote machines.

If an array is on a private network with trusted peers, or perhaps is carefully hidden behind a good firewall, authentication may be unnecessary. It is possible to disable authentication entirely by using an authentication information entry of the form

```
AUTHENTICATION NONE
```

This is the default setting when the array services are first installed. However, unless the environment is reasonably secure, this should be changed to one of the other authentication settings as soon as possible.

WARNINGS

The UNICOS operating system is dependent on the `nobody` user being configured in order to use array services and the message passing interface (MPI).

SEE ALSO

`arrayd`(8) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

`arshell`(1)

`array_services`(7), `array_sessions`(7)

NAME

bld – Relocatable library files format

IMPLEMENTATION

Cray PVP systems only

DESCRIPTION

When the bld(1) command collects relocatable modules, it creates a relocatable library file called a *build file* or a bld *file*. The *build file* consists of a header table, a collection of relocatable modules, and a contents table (also called a *termination table*).

The header table precedes the relocatable modules; it has the following format:

```
struct  bld_hdr {
        struct  tbl_hdr  hdr;
        long    pdt_offset;    /* file offset to the build     */
                               /* termination table            */
                               /* (1 = no pdt entries)         */
        long    pdt_size;      /* size (in words) of the build */
                               /* termination table            */
                               /* (1 = no pdt entries)         */
};
```

The contents table follows the relocatable modules; it consists of a table header followed by copies of all Program Descriptor tables (PDTs) that occur in the modules within the relocatable library. (See relo(5) for descriptions of the table header and PDTs.) The ld(1) and segldr(1) loader commands use the build header table and contents table.

The bld(1) command uses the pdtscl field in each PDT in the build contents table to store a file pointer to the associated module.

FILES

/usr/src/cmd/bld/bld.h	Format of relocatable library files

SEE ALSO

ar(5), relo(5)

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

NAME

`confval` – Configuration file for various products

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `confval` file contains configuration information for various products in the following format:

product`.`*field* `:` *value* `[`*value* `...]`

product Name of the product

field Product-specific field identifier

value Text string (or list of text strings, separated by white space) that the product expects to see for the given configuration field

You must separate the *product* and *field* strings by using a period (`.`), and you must use a colon (`:`) to separate the *field* and first *value* strings. The backslash continuation character (\) is honored if it is immediately followed by a newline character (`\n`).

Any line that starts with a # symbol is considered a comment line and is ignored. Blank lines also are ignored.

To delimit the starting and ending locations of a value, use the `"` symbol; however, new lines are not allowed within a delimited value.

Options:

`login.deflbl_as_minlbl`
: This is a UNICOS centralized user Identification/Authentication (I/A) option for determining a user's mandatory access control (MAC) attributes. If this option is selected, the user's default label also will be used as the user's minimum label. This provides the ability to define a user's minimum compartment set. `ia_mlsuser`(3C) processes this option.

`login.logbadpass`
: This is a UNICOS centralized user Identification/Authentication (I/A) failure processing option. If this option is selected, the failed I/A attempts are logged in the syslog by `ia_failure`(3C). This configuration option is only for systems that have `SECURE_SYS` configured off. `ia_failure`(3C) processes this option.

CAUTIONS

If you edit this file on a running (multiuser) system, binary files may not detect the new configuration information because of the internal buffering of data performed by `getconfval`(3C). For best results, restart the affected binary file and/or binary files.

EXAMPLES

A partial example of a `/etc/config/confval` file follows:

```
#  Partial example for gated(8) configuration
#
gated.debug:    1
gated.rip:      quiet
gated.static:   "128.162.82.124 rip metric 1 active"

#
#  Partial example login(8) configuration, set so that:
#    1. Causes user’s default label to be used as both the default and minimum
#       label for all UDB references  for user’s minimum label (not just login)
#    2. Failed login attempts are not put into the system log
#    3. The user has unlimited attempts during a connection to try to log in

login.deflbl_as_minlbl:  1
login.logbadpass:        0
login.login_attempts:    0
```

SEE ALSO

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

NAME

`core` – Core file format

SYNOPSIS

```
#include <sys/types.h>
#include <sys/param.h>
#include <sys/user.h>
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

A core file is the image of a terminated process; the UNICOS system writes out a core file when various errors occur. The most common errors are memory violations, illegal instructions, and user-generated quit signals; see `signal`(2) for a complete list of possible errors. The process image is written to a file called `core` in the working directory of the process if that directory is writable or if a core file already exists and is writable. If extended core file naming is turned on, the process image is written to a file named `core.`*pid* in the working directory of the process if that directory is writable. A process with an effective user ID different from the real user ID does not produce a core file. See `setuid`(2) for more information on setting user and group IDs.

The core image has two sections. The first section of the image contains the user common structure, `ucomm`, which is of size `UCSIZE` (in clicks) (on Cray Research systems, a *click* is 4096 bytes). It is described in the `sys/param.h` include file. The `ucomm` structure is followed by one or more user structures; each user structure is size `ULSIZE` (in clicks). `USIZE` is still available for compatibility.

The format of a user structure also is described in the `sys/user.h` include file. When the process is not multitasked, exactly one user structure exists; when it is multitasked, one user structure exists for each process (task). The number of user structures in the `core` file is specified by the `uc_core` variable in the `ucomm` structure. The user structures start at offset `UCSIZE` clicks in the `core` file and continue for `uc_usoff` clicks; each user structure has a flag in the user structure, `u_active`, set to a nonzero value if the user structure is in use.

The second section of the image is the user memory area. The second section of the core image is written only when the size of the process is less than the core file size limit, as defined in the `pc_corelimit` field in `sys/proc.h`. (The core file size limit for each user defaults to `unlimited`, but might have been reduced by the system administrator using the `ue_pcorelim` field in the user database (UDB) or by the user using the `limit`(1) command with the `-d` option. For information about determining the core file size limits, see the `limit`(1) man page.) If the attempt to write a complete restartable core file fails, an attempt is made to write a truncated core file, in which only the first section of the core image is written.

Only the data area is dumped if the instruction area is separate from the data area (this is called *split I&D* or *shared text*).

NOTES

The `crash`(8) command can write a core file.

FILES

`/usr/include/sys/param.h`	System parameter file
`/usr/include/sys/proc.h`	Format of the process common structure
`/usr/include/sys/types.h`	Data type definition file
`/usr/include/sys/user.h`	Format of the user common structure

SEE ALSO

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

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

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

NAME

cpio – cpio archive file format

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The cpio archive file is the output from the cpio(1) command, which collects files into an archive. Each file within the archive is preceded by a header that has two possible formats. When you omit the -c option of cpio(1), the header structure is as follows:

```
struct header {
       int    h_magic,
              h_dev;
       uint   h_ino,
              h_mode,
              h_uid,
              h_gid;
       int    h_nlink,
              h_rdev;
       int    h_param[8];
       long   h_mtime;
       int    h_namesize;
       long   h_filesize;
       char   h_name[h_namesize rounded to word];
}
```

When the -c option of cpio(1) is used, the header information is described by the following:

```
sscanf(Chdr,"%6o%6o%6o%6o%6o%6o%6o%6o%11lo%6o%11lo%s",
       &Hdr.h_magic, &Hdr.h_dev, &Hdr.h_ino, &Hdr.h_mode,
       &Hdr.h_uid, &Hdr.h_gid, &Hdr.h_nlink, &Hdr.h_rdev,
       &Longtime, &Hdr.h_namesize, &Longfile, Hdr.h_name);
```

Longtime and Longfile are equal to Hdr.h_mtime and Hdr.h_filesize, respectively. The contents of each file immediately follow the archive header that describes the file.

If h_aclcount is nonzero, the access control list (ACL) entries immediately precede the header for the following file.

Each instance of h_magic contains the constant 070707 (octal). Items h_dev through h_mtime correspond to the items in the stat structure explained in stat(2). The length of the null-terminated path name h_name, including the null byte, is given by h_namesize.

The last record of the cpio archive always contains the name TRAILER!!!. Special files, directories, and the trailer are recorded with h_filesize equal to 0.

For a cpio archive that contains security labeling, the -c option is not allowed. The following header structure precedes the previously described header structure for each archived file:

```
/* Secure cpio header format */
struct sheader {
       int     h_smagic;
       int     h_slevel;
       long    h_compart;
       long    h_acldsk;
       int     h_aclcount;
       long    h_hdrvsn;
}
```

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

The following secondary security header structure immediately follows the sheader structure:

```
/*
 *  Additional cpio secure header
 */
struct nheader {
       int     h_nmagic;
       int     h_intcls;
       long    h_intcat;
       long    h_secflg;
       int     h_minlvl;
       int     h_maxlvl;
       long    h_valcmp;
       long    h_reserved[16];
}
```

Each instance of h_nmagic contains the constant 050505 (octal).

If PHdr is in the archive, the first item in the archive is the PHdr. The PHdr contains the privilege authorization list (PAL) header. The PAL header has the following structure and a magic number of 040404:

```
struct      pheader {
                int     h_pmagic;
                pal_t   h_pal;
}PHdr;
```

SEE ALSO

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

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

NAME

cshrc, login, logout – C shell start-up and termination files

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

If you specify /bin/csh in the shell field of the password file, the following actions occur as a user logs in:

1. If the /etc/cshrc file exists, the C shell (csh(1)) executes it. Among other operations, /etc/cshrc prints /etc/motd, the message of the day, if that file exists (see motd(5)).
2. If the user's login directory contains a file named .cshrc, csh(1) executes it.
3. If the user's login directory contains a file named .login, csh(1) executes it.
4. The user's terminal session begins.

Files /etc/cshrc and .login are executed only on login, but file .cshrc is executed each time csh(1) is executed. Therefore, .login is useful for setting and exporting environment variables and for executing commands desired on login (for example, calendar(1)); .cshrc is useful for setting up aliases and other environment parameters that should be set each time csh(1) is executed.

When a login C shell terminates, the $HOME/.logout file is executed. The user or system administrator creates the .logout file, which contains commands to be executed on shell termination. For example, a .logout file might include commands to clear the screen and to erase temporary files.

EXAMPLES

Example 1: An example of a typical .login file is as follows:

```
# Set file creation mask:
umask 22
# Echo a greeting:
echo "Welcome to the Cray Research computer system"
# Establish command search path
setenv path=($PATH   $HOME/bin)
```

Example 2: An example of a typical `.cshrc` file is as follows:

```
# Check for interactive mode and set prompt and history:
if  (  $?prompt  )  then
     set prompt  =  "CRAY>  "
     set history  =  22
endif

# Set some aliases:
alias l     ls -al
alias h     history -r
```

Example 3: An example of a typical `.logout` file is as follows:

```
# Remove files in personal temporary directory
rm $HOME/tmp/*
# Clear the screen
clear
```

FILES

`$HOME/.cshrc`	C shell start-up file in user's home directory
`$HOME/.login`	C shell start-up file in user's home directory
`$HOME/.logout`	C shell termination file in user's home directory
`/bin/csh`	`csh` command
`/etc/cshrc`	Systemwide C shell start-up file
`/etc/udb`	User information file

SEE ALSO

`motd`(5), `profile`(5)

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

NAME

def_seg, def_ld, ld_Flib – Loader directives files

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The /lib/segdirs/def_seg, /lib/segdirs/def_ld, and /lib/segdirs/ld_Flib files are loader directives files. The default directives files contain initial information used for the loading process. This information includes machine-specific program construction data, the library names (if any) that are searched by default, and the location of the libraries to be searched.

The /lib/segdirs/def_seg file is the default directives file for the segldr(1) command; /lib/segdirs/def_ld is the default directives file for the ld(1) command. When either loader command begins execution, it reads the contents of the default directives file for that command.

When you specify the -F option on the ld(1) command line, the ld(1) command uses the /lib/segdirs/ld_Flib file. It describes the libraries ld(1) should search when flowtracing has been enabled or when a user wants to include the complete set of default libraries. This file should identify the same libraries that the def_seg file specifies for segldr.

The initial contents of these files are created when the system is installed. To customize the loader actions, the system administrator can add or remove directives in any of the files.

EXAMPLES

The following examples show the contents of the three loader directives files.

Sample def_seg file:

```
system=unicos        /* set the target operating system            */
start=$START         /* declare the name of the program entry point */
callxfer=M$A$I$N     /* declare the name of the transfer reference  */
compress=1000        /* declare the compression threshold value     */
hardref=trbk         /* force hard references to entry 'trbk'       */
deflib=libc.a        /* identify the default libraries              */
deflib=libu.a
deflib=libm.a
deflib=libf.a
deflib=libio.a
deflib=libsci.a
deflib=libp.a
```

Sample `def_ld` file:

```
system=unicos        /* set the target operating system            */
start=$START         /* declare the name of the program entry point */
callxfer=M$A$I$N     /* declare the name of the transfer reference  */
compress=1000        /* declare the compression threshold value     */
lbin=_start_.o       /* load the system start-up routine first      */
```

Sample `ld_Flib` file:

```
deflib=libc.a        /* identify the default libraries              */
deflib=libu.a
deflib=libm.a
deflib=libf.a
deflib=libio.a
deflib=libsci.a
deflib=libp.a
```

FILES

`/lib/segdirs/def_ld`	Default directives file for `ld`
`/lib/segdirs/def_seg`	Default directives file for `segldr`
`/lib/segdirs/ld_Flib`	Identifies libraries used by `ld`

SEE ALSO

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

NAME

dir, ncdir – Directory file format

SYNOPSIS

```
#include <sys/fs/ncdir.h>
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

A directory functions as a regular file, except that no user may write to a directory. The mode field of a file's inode entry indicates whether a file is a directory.

```
struct cdirect
{
       unsigned long  cd_ino;               /* Inode for name           */
       unsigned long  cd_sino;              /* Reserved for future use  */
       unsigned short cd_reserved:10,       /* Reserved for future use  */
                      cd_signature:22,      /* Name signature           */
                      cd_reclen:22,         /* Record length (bytes)    */
                      cd_namelen:10;        /* Length of name (bytes);  */
       unsigned char  cd_name[CDMAXNAMELEN]; /* Directory name          */
} ;
```

By convention, the first two entries in each directory are "." and "..". The first is an entry for the directory itself, and the second is for the parent directory. The meaning of ".." is modified for the root directory of the master file system; because no parent directory exists, ".." has the same meaning as ".".

An unused directory entry, identified by cd_ino = cd_namelen = 0, is permitted only at the beginning of a block.

Directory names are null-padded to the nearest word boundary. If the name length is a multiple of 8, a null-terminator is not guaranteed.

FILES

/usr/include/sys/dir.h	Not used; retained for compatibility.
/usr/include/sys/fs/ncdir.h	NC1FS file systems.

SEE ALSO

dirent(5), fs(5)

NAME

dirent – File system-independent directory entry format

SYNOPSIS

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

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

Different file system types may have different directory entries. The dirent structure defines a file system-independent directory entry, which contains information common to directory entries in different file system types. The getdents(2) system call returns a set of these structures; you can access these structures by using the closedir(3C), opendir(3C), readdir(3C), rewinddir(3C), seekdir(3C), and telldir(3C) routines (see directory(3C)).

The dirent structure is defined as follows:

```
struct  dirent {
        long           d_ino;
        off_t          d_off;
        unsigned short d_reclen;
        char           d_name[1];
};
```

d_ino — Unique number for each file in the file system.

d_off — Offset from the beginning of the file to the end of the current entry.

d_reclen — Record length of the entry; defined as the number of bytes required between the current entry and the next one to ensure that the next entry is on a word boundary.

d_name — Beginning of the character array that gives the name of the directory entry. This name is null-terminated and has a maximum character length of MAXNAMLEN characters. This results in file system-independent directory entries being variable-length entities.

FILES

/usr/include/sys/dirent.h	File system-independent directory entry definition file
/usr/include/sys/types.h	Data type definition file

SEE ALSO

`dir`(5), `fs`(5)

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

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

NAME

dump, dumpdates – Incremental file system dump format

SYNOPSIS

```
#include <sys/types.h>
#include <sys/param.h>
#include <sys/fs/nc1ino.h>
#include <dumprestor.h>
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

File system dump tapes used by the dump(8) and restore(8) commands contain the following information:

- A header record
- Two groups of bitmap records
- A group of records that describes directories
- A group of records that describes nondirectory files
- A trailing bitmap

The following symbols are defined in dumprestor.h, (the entries prefaced with TS_ are used in the c_type field to indicate the header type):

```
#define CNTREC    8
#define TS_TAPE   1
#define TS_INODE  2
#define TS_BITS   3
#define TS_ADDR   4
#define TS_END    5
#define TS_CLRI   6
#define TS_ACL    7
#define TS_PAL    8
#define DR_MAGIC  (int) 60012
#define CHECKSUM  (int) 84446
```

CNTREC Number of 4096-byte records in a physical tape block.

TS_TAPE First block of dump output.

TS_INODE File or directory follows. The c_dinode field is a copy of the disk inode; it contains bits that specify the type of the file.

`TS_BITS` Bit map follows. This bitmap consists of 1 bit for each inode that was dumped. At the end of the dump output, a second `TS_BITS` bitmap indicates the inodes that were updated during execution of `dump`.

`TS_ADDR` A subrecord of a file description, (see the description of `c_addr`).

`TS_END` End-of-tape record.

`TS_CLRI` Bit map follows. This bitmap contains a 0 bit for all inodes that were empty when the file system was dumped.

`TS_ACL` Access control list (ACL) block follows.

`TS_PAL` Privilege assignment list (PAL) block follows.

`DR_MAGIC` A magic number.

`CHECKSUM` Checksum for header records.

Header Record Format

The `dumprestor.h` include file defines the format of the header record and of the first record of each description.

```
union cu_spcl {
        char dummy[BSIZE];
        struct  c_spcl {
                int     c_type;
                time_t  c_date;
                time_t  c_ddate;
                long    c_tapea;
                long    c_inumber;
                int     c_checksum;
                struct  cdinode c_dinode;
                int     c_count;
                char    c_addr[NINDIR];
        } c_spcl;
};

#define cspcl cu_spcl.c_spcl
```

`c_type` Header type.

`c_date` Date of dump.

`c_ddate` Date of previous incremental dump.

`c_tapea` Current number of this 4096-byte record.

`c_inumber` Number of inode being dumped if `TS_INODE` is set.

`c_checksum` The value needed to make the record's checksum equal to `CHECKSUM`.

`c_dinode` Copy of inode as it appears in the file system; for a description of the inode format, see `fs`(5).

`c_count` Count of characters in `c_addr`.

`c_addr` Array of characters that describes the blocks of the dumped file, 1 bit per character. If the block associated with that character was not present on the file system when it was dumped, a character is 0; otherwise, the character is nonzero. If the block was not present on the file system, the block will be restored as a hole in the file. If there is not sufficient space in this record to describe all of the blocks in a file, `TS_ADDR` subrecords are scattered throughout the file, each one starting where the last one left off.

Dump History

The dump history is kept in the `/etc/dumpdates` file. The format of an entry in `/etc/dumpdates` is as follows:

name level date(*timestamp*) *volume* [`:` *volume*]

name Name of dumped file system.

level Level number of dump tape (see `dump`(8)).

date Date of incremental dump in `date`(1) format.

timestamp Date of the incremental dump in seconds since 00:00:00 GMT, January 1, 1970.

volume Volume serial number of the dump tape; if the dumped file system is contained on more than one tape, the numbers are separated by colons (`:`). If the file system was not dumped to a tape, the word `NULL` appears in this field.

To specify this field, use the `-T` option on the `dump`(8) command. The default is the first 40 characters of the VSN list.

FILES

`/etc/dumpdates`	Incremental file system dump file
`/usr/include/dumprestor.h`	File system dump tape header definition
`/usr/include/sys/inode.h`	Inode structure definition
`/usr/include/sys/types.h`	Data type definition file

SEE ALSO

`fs`(5), `types`(5)

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

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

NAME

errfile – Error-log file format

SYNOPSIS

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

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

When the system detects hardware errors, an error record is generated and passed to the error-logging daemon, errdemon(8). The error-logging daemon records the error record in the error-log file for later analysis. The default error-log file is /usr/adm/errfile.

The format of an error record in an error-log file depends on the type of error encountered. Each record, however, has a header with the following format defined in the sys/erec.h include file:

```
struct errhdr {
       short  e_type;        /* record type                          */
       short  e_len;         /* bytes in record (with header)        */
       time_t e_time;        /* time of day                          */
} ;
```

The permissible record types are as follows:

```
#define E_GOTS 010          /* Start for UNICOS/TS                     */
#define E_STOP 012          /* Stop                                    */
#define E_TCHG 013          /* Time change                             */
#define E_SSD    0100       /* SSD error record                        */
#define E_MEM    01000      /* CRAY memory error                       */
#define E_SDIS   01001      /* Single-bit error detection disabled */
#define E_SEN    01002      /* Single-bit error detection enabled  */
#define E_IOS    01003      /* IOS error packet(s)                     */
#define E_DSK    01004      /* Disk driver error report                */
#define E_D29    01005      /* DD29 error record                       */
#define E_D39    01006      /* DD39 error record                       */
#define E_D49    01007      /* DD49 error record                       */
#define E_D40    01010      /* DD40 error record                       */
#define E_D10    01011      /* DD10 error record                       */
#define E_D50    01012      /* DD50 error record                       */
#define E_D11    01013      /* DD11 error record                       */
#define E_D41    01014      /* DD41 error record                       */
#define E_TAPE   01021      /* Tape error record                       */
#define E_PARITY 01030      /* Register parity                         */
#define E_HIPPI  01050      /* HIPPI error                             */
```

`E_GOTS` Error-logging start-up record; when logging is first activated, one of these is sent to the error-logging daemon.

`E_STOP` Error-logging termination record; when it stops logging errors, one of these is sent to the daemon.

`E_TCHG` Time-change record; whenever the system's time of day is changed, one of these is sent to the daemon.

`E_SSD` SSD error record; one of these is generated for each SSD error.

`E_MEM` Memory error record; one of these is generated for each memory error.

`E_SDIS` Marker record signifying that single-bit error detection is disabled.

`E_SEN` Marker record signifying that single-bit error detection is enabled.

`E_IOS` IOS error record.

`E_DSK` Disk error record.

`E_D29` DD-29 error record.

`E_D39` DD-39 error record.

`E_D49` DD-49 error record.

`E_D40` DD-40 error record.

`E_D10` DD-10 error record.

`E_D50` DD-50 error record.

`E_D11` DD-11 error record.

`E_D41` DD-41 error record.

`E_TAPE` Tape error structure.

`E_PARITY` Register parity error record.

`E_HIPPI` HIPPI error record.

The error file contains some administrative records. These include `E_GOTS` (the start-up record entered into the file when logging is activated), `E_STOP` (the record written when the error-logging daemon is terminated gracefully), and `E_TCHG` (the time-change record that accounts for changes in the system's time of day). The formats for these records are defined in the `sys/erec.h` include file.

FILES

`/usr/adm/errfile`	Default error-logging file
`/usr/include/sys/erec.h`	Error-log header format

SEE ALSO

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

NAME

`exports`, `xtab` – Directories to export to NFS clients

SYNOPSIS

`/etc/exports`
`/etc/xtab`

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `/etc/exports` file contains entries for directories that can be exported to NFS clients. The `exportfs`(8) command reads this file automatically. If you change this file, you must run `exportfs`(8) for the changes to affect the daemon's operation.

The `/etc/xtab` file contains entries for directories that are currently exported. (To remove entries from this file, use the `-u` option of `exportfs`(8).)

An entry for a directory consists of a line that has the following format:

directory `-`*option*`[,`*option*`]...`

directory Path name of a directory

The following are valid options:

`ro` Exports the directory read-only. If you omit this option, the directory is exported read-write.

`rw=`*hostname*`[:`*hostname*`]...`
Exports the directory read-mostly. *Read-mostly* means read-only to most hosts, but read-write to the hosts that are specified by *hostname*. If you omit this option, the directory is exported read-write to all. `ro` and `rw` are mutually-exclusive options.

`anon=`*uid* Specifies *uid* as the effective user ID when a request comes from an unknown user.

Users who are logged in as root (*uid 0*) are always considered unknown by the NFS server, unless they are included in the root option that follows. The default value for the `anon` option is –2. To disable anonymous access, set `anon = -1`.

`root=`*hostname*`[:`*hostname*`]...`
Gives root access only to the root users from a specified host. The default is that no hosts are granted root access.

`access=`*client*`[:`*client*`]...`
Gives mount access to each client listed. The default value allows any machine to mount the given directory.

`cksum` Checksums packets that are returned to clients.

`krb` Indicates that Kerberos authentication is required for access to this export.

`nosync` Specifies that write operations to this file system are delayed. This option can significantly improve write performance, but its use can cause loss of data if the server crashes before the data is written to disk.

A # symbol anywhere in the file indicates a comment, which extends to the end of the line.

The *client* argument can specify the name of a host or the name of a netgroup. For information on how to use a netgroup file, see `netgroup`(5).

CAUTIONS

You cannot export either a parent directory or a subdirectory of an exported directory that is within the same file system. When both directories reside on the same disk partition, it is illegal, for example, to export both `/usr` and `/usr/local`.

EXAMPLES

An example of an `exports` file follows:

```
/usr        -access=clients           # export to my clients
/usr/local                            # export to the world
/usr2       -access=hermes:zip:aspen  # export to only these machines
/usr/sun    -root=hermes:zip          # give root access only to these
                                      # hosts
/usr/new    -anon=0                   # give all machines root access
/usr/bin    -ro                       # export read-only to everyone
/usr/stuff  -access=zip,anon=-3,ro    # several options on one line
/usr/other  -rw=host1:host2:host3     # read-write to listed hosts
```

FILES

`/etc/exports` Contains a list of directories that are exportable to NFS clients

`/etc/hosts` Contains a list of known hosts on a network

`/etc/xtab` Contains a list of directories that are currently exported

SEE ALSO

`hosts`(5), `netgroup`(5)

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

NAME

exrc – Start-up files for ex(1) and vi(1)

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The .exrc file is a start-up file for the ex(1) and vi(1) text editors. In this file, you can enter editor commands that you want the editor to execute every time you edit a file. When you invoke ex(1) or vi(1), the editor checks for a file named .exrc in your home directory ($HOME) and runs editor commands it finds there. Then the editor checks for a file named .exrc in the current directory. The .exrc file in the current directory is, by default, processed only if you are the owner. This default can be changed only to allow the processing of .exrc files that you do not own at the time the editor command is built.

You can enter each editor command on a separate line or enter several separate commands on the same line and separate the commands with a | symbol.

NOTES

If you have the EXINIT environment variable defined, the .exrc files are not processed.

EXAMPLES

Three useful editor commands that are commonly included in .exrc files are shown in the following examples. For a complete description of all editor commands, see ex(1) or vi(1).

Example 1: In the following example, the file contains the set command twice with two separate options. The first option turns off wrapscan, so that when a search is in progress, the editor does not wrap to the beginning when the end of the file has been reached. The second option sets showmatch, which tells the editor to show the match to a right parenthesis ( )) or right brace (}) when either of these characters is entered.

```
set nowrapscan | set showmatch
```

Example 2: To replace a keystroke entered in the vi(1) command mode with a series of vi(1) commands, use the map command. The following example uses map to set the "=" character to the vi(1) command 5x. When a user types the string =, vi(1) executes 5x, deleting 5 characters.

```
map  =  5x
```

Example 3: The vi(1) command, abbreviate, is specified in the following example. When this command is in effect, vi(1) automatically replaces string "cri" with "Cray Research, Incorporated" when you type the string in insert mode. (This substitution occurs only when "cri" is surrounded by spaces, tabs, or punctuation; this ensures that substitutions do not occur in the middle of words.)

```
abbreviate cri Cray Research, Incorporated
```

FILES

`$HOME/.exrc`	Editor start-up file in your home directory
`./.exrc`	Editor start-up file in the current directory

SEE ALSO

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

NAME

fcntl – File control options

SYNOPSIS

```
#include <fcntl.h>
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The fcntl(2) system call provides control over open files. The fcntl.h include file describes the available requests and arguments to fcntl(2) and open(2).

```
/* Flag values accessible to open(2) and fcntl(2) */
/*  (The first three can only be set by open)     */

#define O_RDONLY   0
#define O_WRONLY   1
#define O_RDWR     2
#define O_NDELAY   04      /* Non-blocking I/O                        */
#define O_APPEND   010     /* append (writes guaranteed at the end)   */
#define O_SYNC     020     /* synchronous write option                */
#define O_NONBLOCK 040     /* Non-blocking I/O (POSIX)                */
#define O_RAW      0100    /* direct to user space                    */
                           /* (no system buffering) I/O               */
#define O_SSD      0200    /* I/O addresses are SDS relative          */
                           /* (CRAY Y-MP systems) */
#define O_ASYNC    0200000 /* Asynch I/O: for sockets                 */

/* Flag values accessible only to open(2) */

#define O_CREAT    000400  /* open with file create                   */
                           /* (uses third open arg)                   */
#define O_TRUNC    001000  /* open with truncation                    */
#define O_EXCL     002000  /* exclusive open                          */
#define O_NOCTTY   004000  /* No controlling TTY (POSIX)              */
#define O_RESTART  040000  /* create file as a restart file           */

/* fcntl(2) requests */

#define F_DUPFD     0      /* Duplicate fildes                        */
#define F_GETFD     1      /* Get fildes flags                        */
#define F_SETFD     2      /* Set fildes flags                        */
```

```
#define F_GETFL     3      /* Get file flags                          */
#define F_SETFL     4      /* Set file flags                          */
#define F_GETLK     5      /* Get file lock                           */
#define F_SETLK     6      /* Set file lock                           */
#define F_SETLKW    7      /* Set file lock and wait                  */
#define F_CHKFL     8      /* Check legality of                       */
                           /* file flag change                        */
                           /* Internal use only                       */
#define F_GETOWN    9      /* Get SIGIO/SIGURG proc/pgrp              */
#define F_SETOWN   10      /* Set SIGIO/SIGURG proc/pgrp              */

/* file segment locking set data type                                 */
/*      - information passed to system by user                        */

struct flock {
       short       l_type;
       short       l_whence;
       long        l_start;
       long        l_len;  /* len = 0 means until end of file         */
       short       l_sysid;
       short       l_pid;
};

/* file segment locking types */

#define F_RDLCK    01      /* Read lock                               */
#define F_WRLCK    02      /* Write lock                              */
#define F_UNLCK    03      /* Remove lock(s)                          */
```

FILES

`/usr/include/fcntl.h` File control options file

SEE ALSO

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

NAME

`fs` – File system partition format

SYNOPSIS

```
#include <sys/types.h>
#include <sys/param.h>
#include <sys/map.h>
#include <sys/fs/nc1filsys.h>
#include <sys/fs/c1filsys.h>
```

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

The `NC1FS` file system is created by The `mkfs`(8) command. For further information about the file system structure of the `NC1FS` file system, see *General UNICOS System Administration*, Cray Research publication SG–2301.

The format of the file system blocks for the `NC1FS` file system is as follows:

```
/*
 * Inode region descriptor.
 * The first block of an inode region is a
 * bit map for the inodes in that region.
 */

struct nc1ireg_sb        {
        uint    i_unused:16,    /* reserved                             */
                i_nbl   :16,    /* number of blocks                     */
                i_sblk  :32;    /* start block number                   */
};

struct nc1ireg_db        {
        uint    i_avail;        /* number of available inodes           */
};
#define NC1MAXIREG      4       /* Maximum inode regions per partition    */
#define NC1IMAPBLKS     1       /* number of blocks in inode map          */

struct      nc1fdev_sb
{
        long            fd_name;                /* Physical device name   */
        uint            fd_sblk :32,            /* Start block number     */
                        fd_nblk :32;            /* Number of blocks       */
```

```
        struct nc1ireg_sb  fd_ireg[NC1MAXIREG]; /* Inode regions          */
};

struct      nc1fdev_db
{
        int                fd_flag;              /* flag word             */
        struct nc1ireg_db  fd_ireg[NC1MAXIREG];  /* Inode regions         */
};

#define FDNC1_DOWN      01      /* Slice not available                      */
#define FDNC1_RDONLY    2       /* Slice is read only                       */
#define FDNC1_NOALLOC   4       /* Slice is not available for allocation    */
#define FDNC1_SBDB      010     /* Slice has valid FS tables                */
#define FDNC1_RTDIR     020     /* Slice has valid ROOT Inode and directory */
#define FDNC1_SECALL    0100    /* Slice sector allocated                   */

#define NC1MAXPART      64      /* Maximum number of partitions             */


/*
 * Structure of the super-block
 */

struct  nc1filsys
{
        long    s_magic;        /* magic number to indicate file system type  */
        char    s_fname[8];     /* file system name                            */
        char    s_fpack[8];     /* file system pack name                       */
        dev_t   s_dev;          /* major/minor device, for verification       */

        daddr_t s_fsize;        /* size in blocks of entire volume            */
        int     s_isize;        /* Number of total inodes                      */
        long    s_bigfile;      /* number of bytes at which a file is big     */
        long    s_bigunit;      /* minimum number of blocks allocated for
                                   big files                                    */
        long    s_secure;       /* security: secure FS label                  */
        int     s_maxlvl;       /* security: maximum security level           */
        int     s_minlvl;       /* security: minimum security level           */
        long    s_valcmp;       /* security: valid security compartments      */
        time_t  s_time;         /* last super block update                     */
        blkno_t s_dboff;        /* Dynamic block number                        */
        ino_t   s_root;        /* root inode                                   */
        struct  nc1dblock *s_pdb; /* pointer to dynamic block (when mounted)  */
        blkno_t s_mapoff;       /* Start map block number                      */
```

```
        int     s_mapblks;      /* Last map block number                       */
        int     s_nscpys;       /* Number of copies of s.b per partition       */
        int     s_npart;        /* Number of partitions                        */
        int     s_ifract;       /* Ratio of inodes to blocks                   */
        blkno_t s_sfs;          /* reserved                                    */
        long    s_flag;         /* Flag word                                   */

        struct  nc1fdev_sb s_part[NC1MAXPART];  /* Partition descriptors      */
        int     s_iounit;       /* Physical block size                         */
        long    s_numiresblks;  /* number of inode reservation blocks          */
                                /* per region (currently 1)                    */
                                /* 0 = 1*(AU) words, n = (n+1)*(AU) words      */
        long    s_priparts;     /* bitmap of primary partitions                */
        long    s_priblock;     /* block size of primary partition(s)          */
                                /* 0 = 1*512 words, n = (n+1)*512 words        */
        long    s_prinblks;     /* number of 512 wds blocks in primary         */
        long    s_secparts;     /* bitmap of secondary partitions              */
        long    s_secblock;     /* block size of secondary partition(s)        */
                                /* 0 = 1*512 words, n = (n+1)*512 words        */
        long    s_secnblks;     /* number of 512 wds blocks in secondary       */
        long    s_sbdbparts;    /* bitmap of partitions with file system data */
                                /* including super blocks, dynamic block       */
                                /* and free block bitmaps (only primary        */
                                /* partitions may contain these)               */
        long    s_rootdparts;   /* bitmap of partitions with root directory    */
                                /* (only primary partitions)                   */
        long    s_nudparts;     /* bitmap of no-user-data partitions           */
                                /* (only primary partitions)                   */
        long    s_fill[94];     /* reserved                                    */
};

struct      nc1dblock
{
        long    db_magic;       /* magic number to indicate file system type  */
        daddr_t db_tfree;       /* total free blocks                           */
        int     db_ifree;       /* total free inodes                           */
        int     db_ninode;      /* total allocated inodes                      */
        long    db_state;       /* file system state                           */
        time_t  db_time;        /* last dynamic block update                   */
        long    db_type;        /* type of new file system                     */
        int     db_spart;       /* Partition from which system mounted         */
        int     db_ifptr;       /* Inode allocation pointer                    */
        int     db_actype;      /* device accounting type (for billing)        */
        long    db_flag;        /* Flag word                                   */
```

```
        long    db_res1;        /* reserved                                      */
        struct  buf *db_fbuf;   /* Free block map buffer descriptor              */
        struct  map db_fpmap;   /* Free block map header - primary parts         */

        struct  nc1fdev_db db_part[NC1MAXPART];  /* Partition descriptors        */
        lockinfo_t  db_lockinf; /* proc of the process locking the filesystem */
        int     db_dpfptr;      /* primary partitions allocation pointer         */
        int     db_dsfptr;      /* secondary partitions allocation pointer       */
        daddr_t db_sfree;       /* secondary parts free blocks                   */
        struct  map db_fsmap;   /* Free block map header - secondary parts       */
        long    db_fill[157];   /* reserved                                      */
};
#define db_proc db_lockinf.hi_proc /* proc of process locking filesystem      */
#define db_fptr db_ifptr
#define db_fmap db_fpmap

/*
 * Filesystem flags
 */

#define         Fs_NOSPC        1       /* Filesystem out of space              */
#define         Fs_RRFILE       2       /* Round robin file allocation          */
#define         Fs_RRALLDIR     4       /* Round robin all directories          */
#define         Fs_RR1STDIR     010     /* Round robin 1st level directories    */
#define         Fs_RDONLY       020     /* File system read only                */
#define         Fs_CHECKED      040     /* File system checked                  */
#define         Fs_MOUNTED      0100    /* File system mounted                  */
#define         Fs_WANTED       0200    /* File system lock wanted              */
#define         Fs_LOCKED       0400    /* File system locked                   */
#define         Fs_UPDATE       01000   /* File system update in progress       */
#define         Fs_WUPDAT       02000   /* File system wakeup after update      */
#define         Fs_RRALLUDATA   020000  /* Round robin all user file data       */
#define         Fs_DIRTY        0100000 /* File system dirty                    */
#define         Fs_SFS          010000000  /* File system shared                */


#define FsMAGIC_NC1 0x6e6331667331636e  /* s_magic number                       */
#define DbMAGIC_NC1 0x6e6331646231636e  /* db_magic number                      */

#define FsSECURE 0xcd076d1771d670cd     /* s_secure: secure file system         */

#define NC1NSUPER       10              /* Copies of s.b. per partition         */
#define NC1MINPARTSZ    (6+NC1NSUPER)   /* Minimum blocks per partition         */
```

```
#define NC1DB(fp) \
      ((struct nc1dblock *)(fp->s_pdb))
#define nc1getfs(mp)    \
        ((struct nc1filsys *)(((struct buf *)(mp)->m_bufp)->b_waddr))
#define nc1getdb(mp)    \
        ((struct nc1dblock *)(((struct buf *)(mp)->m_dbufp)->b_waddr))
```

FILES

/usr/include/sys/fs/nc1filsys.h	Format of file system partitions for NC1FS file systems
/usr/include/sys/map.h	Definitions for bit map management
/usr/include/sys/param.h	System parameter file
/usr/include/sys/types.h	Data type definition file

SEE ALSO

dir(5), dirent(5), inode(5)

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

General UNICOS System Administration, Cray Research publication SG–2301

NAME

fslrec – File system error log record format

SYNOPSIS

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

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

File system error log records are written to /dev/fslog by the UNICOS kernel as part of the panic-less file system feature. The file system error log daemon, fslogd(8), reads and processes log records.

The format of each file system error log record is defined as follows:

```
struct fslgrec {
        time_t fsl_time;       /* time (seconds since '70)           */
        int    fsl_type;       /* error type                         */
        int    fsl_subtype;    /* error sub-type                     */
        char   *fsl_ptr;       /* generic pointer to struct in err   */
};
```

The following list summarizes the file system error log record types:

FSLG_GO	(Deferred) File system error log start record
FSLG_STOP	(Deferred) File system error log stop record
FSLG_FS	The UNICOS kernel has detected a file system data structure error
FSLG_DIR	The UNICOS kernel has detected a directory block error
FSLG_INODE	The UNICOS kernel has detected an error in the memory copy of a file inode

FILES

/dev/fslog	File system error log device
/usr/include/sys/fslog.h	File system error log header file
/usr/include/sys/fslrec.h	Format of file system error log record
/usr/include/sys/types.h	Data type definition file

SEE ALSO

`fslog`(4)

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

NAME

fstab, mntent – File that contains static information about file systems

SYNOPSIS

```
#include <mntent.h>
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The /etc/fstab file describes the file systems and swapping partitions that UNICOS uses. The "mount *directory*" form of the mount(8) command uses this information. The command searches the /etc/fstab file for an entry that has a mount point named *directory* and mounts the file system as the entry describes.

File system quota commands also use this information. For more information, see quadmin(8), quota(1), and qudu(8).

The mfsck(8) command also uses this information. For more information, see mfsck(8).

The system administrator creates the fstab file by using a text editor or the UNICOS installation and configuration menu system. The mount(8) command processes it as a source of default options. The /etc/fstab file is not changed by programs; it is only read. The system administrator must properly create and maintain this file.

The controlling agent for mounting root file systems is the /etc/config/rcoptions file, which is defined with the UNICOS installation and configuration menu system and is used at system startup.

The /etc/fstab file consists of several lines of the following form:

filesystem directory type options frequency passnumber

Each line in the file constitutes a file system entry. The entry fields are separated by white space. The mntent structure definition explains the meaning of each field. A # as the first nonwhite character on a line indicates a comment.

The entries in /etc/fstab are accessed using the routines in getmntent(3C), which return a structure that has the following format:

```
struct mntent {
       char *mnt_fsname;   /* file system name             */
       char *mnt_dir;      /* file system path prefix      */
       char *mnt_type;     /* file system type             */
       char *mnt_opts;     /* ro, quota, etc.              */
       int  mnt_freq;      /* dump frequency, in days      */
       int  mnt_passno;    /* pass number on parallel fsck */
};
```

`mnt_fsname` Name of the block special file to be mounted.

`mnt_dir` Directory mount point for the special file.

`mnt_type` Type of file system specified in `mnt_fsname`. Valid types are `NC1FS`, `NFS`, `PROC`, `INODE`, `SFS`, and `ignore`. If the `mnt_type` is specified as `ignore`, the entry is ignored. This is useful for showing disk partitions that are currently unused.

`mnt_opts` String of comma-separated options. The description of the `fsckopt` and `quota` options follow, but the other options are documented with `mount`(8).

`mnt_freq` Optional field referenced by the `-w` option of the `dump`(8) command to determine the frequency of system dumps.

`mnt_passno` Optional field referenced by the `mfsck`(8) program to determine the order that file systems are checked using the `fsck`(8) command.

`fsckopt` Specifies the file system `mfsck`(8) options when invoking `fsck`(8). This option takes the following form:

`fsckopt=q`

Using `q` as the `fsckopt` specifies that `mfsck`(8) use file system flags to determine when the file system is checked. Specifying `u` as the `fsckopt` implies an unconditional file system check, which is the default.

`quota` Specifies the file quota configuration of the `mnt_opts` entry. This option takes one of three forms:

1. `quota`=*quota_file_relative_name*

   This form is used if the `quota` control file will reside on the file system it controls. The file name is relative to the root directory of the file system, and if the default name is used as recommended, the option generally would be written as `quota=$QFILE`.

   The special name `$QFILE` means the default `quota` file name (as defined in `quadmin`(8)). The default name is `.Quota60` so that the preceding `quota` option would resolve to `quota=.Quota60` in the root directory of the file system.

2. `quota`=*quota_file_full_name*

   This form is used if the `quota` control files will reside in some arbitrary place (for example, if the `quota` files were to reside in the `/etc/admin/quota60` directory, this form could be written as `quota=/etc/admin/quota60/$FILESYS`).

   The special name `$FILESYS` is the last component of the *filesystem* name on this `fstab` line. If this line had been written as

   ```
   /dev/dsk/slash_b /b NC1FS quota=/etc/admin/quota60/$FILESYS
   ```

   it would be resolved to

`quota=/etc/admin/quota60/slash_b`

A directory, `quota60`, was created to hold all of the `quota` control files. The file system name identifies each individual `quota` control file within the directory.

3. `quota=`*/dev/dsk/filesystem_name*

   This form shows that this file system is under the control of a `quota` file defined and used to control another file system. When multiple file systems are controlled as a group, this form is used. For example, assume that three lines from `/etc/fstab` were written as follows:

   ```
   /dev/dsk/tmp_1 /tmp_1 NC1FS quota=$QFILE
   /dev/dsk/tmp_2 /tmp_2 NC1FS quota=/dev/dsk/tmp_1
   /dev/dsk/tmp_3 /tmp_3 NC1FS quota=/dev/dsk/tmp_1
   ```

   These lines define the `quota` control file as `.Quota60` residing in the root directory of `/dev/dsk/tmp_1`. The same `quota` control file controls file systems `/dev/dsk/tmp_2` and `/dev/dsk/tmp_3`; therefore, the `quota` information for usage of any or all of the three file systems is common and reflects the combined usages of all three.

   The rule for using this form is if the right-hand side of a `quota` option matches one of the other file system names in `/etc/fstab`, it is the third form of declaration (as defined previously), and the file system must contain a `quota` option naming a file. Only one level of indirection is supported.

EXAMPLES

File system `/usr/sierra` from remote host `sierra` will be mounted on local directory `/nfs/sierra`. File system type is NFS with options `bg`, `soft`, `rsize`, and `wsize`. For a description of the options, see `mount`(8).

```
sierra:/usr/sierra  /nfs/sierra  NFS  soft,bg,rsize=8192,wsize=8192
```

Mount the `/proc` file system on the `/proc` directory.

```
/proc   /proc   PROC
```

FILES

`/etc/fstab`	File system static information
`/usr/include/mntent.h`	Structure definition of `fstab` entries

SEE ALSO

`mnttab`(5)

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

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

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

NAME

ftpusers – List of unacceptable ftp(1B) users

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The /etc/ftpusers file contains a list of unacceptable ftp(1B) users, one user name per line. When ftp(1B) is run, ftpd(8) checks ftpusers for the login name of the user trying to open a connection. If the user's login name appears in the file, ftpd(8) denies the user access.

If ftpusers is nonexistent or empty, all valid UNICOS users are considered valid users of ftp(1B).

FILES

/etc/ftpusers	File that contains unacceptable ftp(1B) users

SEE ALSO

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

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

NAME

`gated-config` – Gated configuration file syntax

SYNOPSIS

`/etc/gated.conf`

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `gated-config` file consists of a sequence of statements. Statements are composed of tokens separated by white space. Most statements are terminated by a ; symbol. However, directive statements are terminated with a newline. For most statements, white space may contain any combination of blanks, tabs, and newlines; however, directive statements may use only blanks and tabs.

Comments start with a # symbol and run to the end of the line.

There are eight classes of statements in the following list. Statements from the first two classes may occur anywhere in the file.

Class	**Description**
Directive	Specifies included files and the current directory. The parser immediately acts on directives.
Trace option	Controls tracing options.

You must specify the six remaining classes in the following order:

Class	**Description**
Options	Allows specification of some global options.
Interface	Specifies interface options.
Definition	Specifies options, the autonomous system, and martian networks.
Protocol	Enables or disables protocols, and sets protocol options.
Route	Defines static routes by using `static` statements.
Control	Defines routes that are imported from routing peers and routes that are exported to these peers.

Detailed definitions of these classes of statements follow. Primitives that are used in the following definitions are as follows:

host Any host. You can specify a host by its IP address or by a domain name. If you specify a domain name with multiple IP addresses, it is considered an error. The host bits in the IP address must be nonzero.

network Any network. You can specify a network by its IP address or a network name. The host bits in a network specification must be 0. To specify the default network (0.0.0.0), you also may use `default`.

destination Any host or network.

dest_mask Any host or network that has an optional mask. *dest_mask* can be any of the following formats:

`all`
network
network `mask` *mask*
network `mask-length` *bits*
`host` *host*

A mask is a dotted quad that specifies which bits of the destination are significant. To indicate that any IP address can be matched, use `all`. You may use the number of contiguous *bits* instead of an explicit mask.

gateway A host on an attached network.

interface An interface specified by IP address, domain name, or interface name. Be careful with the use of interface names because future UNIX operating systems may allow more than one address per interface.

gateway_list A list of one or more gateways.

interface_list A list of one or more interface names, wildcard names, or addresses, or the token `all`; `all` refers to all interfaces. A wildcard name is an interface name without the number.

preference A number between 0 and 255; 0 is the most preferred, and 255 is the least preferred. *preference* determines the order of routes to the same destination in the routing table. `gated` allows one route to a destination per protocol per autonomous system. For multiple routes, the route to use is chosen by *preference*.

When a *preference* tie exists, if the two routes are from the same protocol and from the same autonomous system, `gated` chooses the route that has the lowest metric. Otherwise, `gated` selects the route with the lowest numeric next-hop gateway address.

metric A valid metric for the specified protocol.

Directives Statements

Directive statements are as follows:

`%directory` *path_name*

Sets the current directory to *path_name*. This is the directory in which `gated` looks for included files that do not begin with a / symbol.

This statement does not actually change the current directory, it simply specifies the prefix applied to included file names.

`%include` *filename*

Causes the specified file to be parsed completely before resuming with this file. Nesting up to 10 levels is supported. To increase the maximum nesting level, change the definition of `FI_MAX` in `parse.h`.

Trace Option Statements

Trace option statements are as follows:

`tracefile` [*filename* [`replace`]] [`size` *size*[`k | m`] `files` *files*];

Specifies the file to contain tracing output. If you specify *filename*, trace information is appended to this file, unless you specify `replace`.

If specified, *size* and *files* cause the trace file to be limited to *size*, with *files* files kept (including the active file). To create the back-up file names, append a period and a number to the trace file name, starting with `.0`. The minimum size that you can specify is 10Kbytes, the minimum number of files that you can specify is 2. The default is not to rotate log files.

`traceoptions` [*traceoption* [*traceoption* [`. . .`]] [`except` *traceoption* [*traceoption* [`. . .`]]];

Changes the tracing options to those specified. If you do not specify any options, tracing is turned off. If you specify the `except` keyword, flags listed before the keyword are turned on, and flags listed after it are turned off. This is a simple method to turn on all but a few flags. Trace flags are as follows:

Flag	Description
`all`	Turns on all of the following options except `nostamp`.
`external`	Produces external error messages.
`general`	Turns on `internal`, `external`, and `route`.
`icmp`	Lists ICMP redirect packets sent and received. To modify it, use `update`. Redirect packets that are processed are traced under the `route` option.
`internal`	Produces internal error and informational messages.
`kernel`	Changes to the kernel's routing table.
`mark`	Indicates that a message will be sent to the trace log every 10 minutes to ensure that `gated`(8) is still running.
`nostamp`	Specifies that all messages in the trace file should not be time-stamped.
`ospf`	Lists OSPF packets sent and received. To modify it, use `protocol`.
`parse`	Lists tokens that the parser recognizes in the configuration file.
`protocol`	Provides messages about protocol state machine transitions when used with `ospf` or `kernel`.
`rip`	Lists RIP packets sent and received. To modify it, use `update`.
`route`	Changes to the `gated`(8) routing table.

`task` Displays task scheduling, signal handling, and packet reception.

`timer` Displays timer scheduling.

`update` Traces the contents of protocol packets.

Options Statements

Options statements are as follows:

`options` *option_list ;*

Sets `gated` options. *option_list* can have these following values:

`noinstall` Does not change the kernel routing table. Useful for verifying configuration files.

`noresolv` Does not try to resolve symbolic names into IP addresses by using the host or network tables or domain name system (DNS). This option is intended for systems in which a lack of routing information could cause a DNS lookup to hang.

`nosend` Does not send any packets. This lets you run `gated`(8) on a live network to test protocol interactions without actually participating in the routing protocols. To verify that `gated`(8) is functioning properly, examine the packet traces in the `gated`(8) log. This is most useful for the RIP protocol.

`syslog` [`upto` *log_level*] *log_level*

Controls the amount of data `gated`(8) logs by using the system log on systems in which the `setlogmask`() routine is supported. The `setlogmask`(3C) man page defines the log levels and other terminology. The default is equivalent to `syslog upto info`.

Interface Statements

Definition statements are as follows:

```
interfaces {
   options [strictinterfaces] [scaninterval time] ;
   interface interface_list interface_options ;
   define address [broadcast broad_addr|pointopoint local_addr ]
      [netmask subnetmask] [multicast] ;
} ;
```

The interface statement includes the following parameters:

`options` Sets some global options related to interfaces. The options are as follows:

`strictinterfaces`

Indicates that it is a fatal error to reference an interface in the configuration file that is not listed in a `define` statement or not present when `gated`(8) is started. Without this option, a warning message is issued and `gated`(8) continues.

`scaninterval` *time*

Specifies how often `gated`(8) scans the kernel interface list for changes. The default is every 15 seconds. `gated`(8) also scans the interface list on receipt of a `SIGUSR2` signal.

`interface` Sets interface options on the specified interfaces. *interface_list* options are as follows:

`all` Specifies the options that apply to all interfaces.

interface_list Specifies a list of interface names, domain names, or numeric addresses. See the warning about interface names in the DESCRIPTION section.

The options are as follows:

`preference` *pref*

Sets the preference for routes to this interface when it is up. The default is 0.

`down preference` *pref*

Sets the preference for routes to this interface when `gated`(8) believes it to be down because of a lack of routing information received. The default is 120.

`passive` Prevents `gated` from changing the preference of the route to this interface if it is believed to be down because of a lack of routing information received.

`define` Defines interfaces that may not be present when `gated`(8) is started. If you specify `strictinterfaces`, `gated`(8) considers it an error to reference a nonexistent interface in the configuration file. This clause allows specification of that interface so that it can be referenced in the configuration file.

Definition keywords are as follows:

`broadcast` *broad_addr*

Defines the interface as broadcast capable (for example, Ethernet and FDDI), and specifies the broadcast address.

`pointopoint` *local_addr*

Defines the interface as a point-to-point interface, and specifies the address on the local side. For this type of interface, the *address* parameter specifies the address of the remote host.

An interface not defined as `broadcast` or `pointopoint` is assumed to be nonbroadcast multiaccess (NBMA), such as HIPPI.

`netmask` *subnetmask*

Specifies the nonstandard subnet mask to be used on this interface. This mask is currently ignored on point-to-point interfaces.

`multicast` Specifies that the interface is multicast-capable.

Definition Statements

Definition statements are as follows:

`autonomoussystem` *autonomous_system*;

Sets the autonomous system of this router to be *autonomous_system*.

`routerid` *host*;

Sets the router identifier for use by the OSPF protocol. The default is the address of the first interface `gated`(8) encounters. The address of a nonpoint-to-point interface is preferred over the local address of a point-to-point interface. The most preferred is an address on a loopback interface that is not the loopback address (127.0.0.1).

`martians {`*martian_list*`};`

Defines a list of martian addresses about which all routing information is ignored. The *martian_list* is a semicolon-separated list of symbolic or numeric hosts that has optional masks. See *dest_mask*. You also may specify the `allow` parameter to allow explicitly a subset of a range that was disallowed.

Protocol Statements

Protocol statements enable or disable use of a protocol and control protocol options. You may specify the protocols in any order. For all protocols, `preference` controls the choice of routes learned through this protocol or from this autonomous system in relation to routes learned from other protocols or autonomous systems.

The default metric used when exporting routes learned from other protocols is specified by using `defaultmetric`, which itself defaults to the highest valid metric for this protocol; for the RIP protocol, this signifies a lack of reachability.

For distance vector protocols (RIP) and redirects (ICMP), the `trustedgateways` clause supplies a list of gateways that provides valid routing information, and routing packets from others are ignored. This defaults to all gateways on the attached networks. Routing packets may be sent not only to the remote end of point-to-point links and the broadcast address of broadcast-capable interfaces, but also to specific gateways if they are listed in a `sourcegateways` clause and `yes` or `on` is specified. If you specify `nobroadcast`, routing updates are sent only to gateways listed in the `sourcegateways` clause, and not to the broadcast address. To disable the transmission and reception of routing packets for a particular protocol, use the `interface` clause. To override an `interface` clause that disables sending or receiving protocol packets for specific peers, use the `trustedgateways` and `sourcegateways` clauses.

Any protocol can have a `traceoptions` clause, which enables tracing for a particular protocol, group, or peer. The allowable protocol-specific options are `all`, `general`, `internal`, `external`, `route`, `update`, `task`, `timer`, `protocol`, or `kernel`.

`rip` Statement

One of the most widely used interior gateway protocols is the Routing Information Protocol (RIP). It classifies routers as active and passive (silent). Active routers advertise their routes (reachability information) to others; passive routers listen and update their routes based on advertisements, but they do not advertise. Typically, routers run RIP in active mode, and hosts use passive mode.

A router running RIP in active mode broadcasts updates at set intervals. Each update contains paired values in which each pair consists of an IP network address and an integer distance to that network. RIP uses a hop count metric to measure the distance to a destination. In the RIP metric, a router advertises directly connected networks at a metric of 1. Networks that are reachable through one other gateway are two hops, and so on. Thus, the number of hops or the hop count along a path from a given source to a given destination refers to the number of gateways that a datagram would encounter along that path.

A RIP routing daemon dynamically builds on information received through RIP updates. When started, it issues a request for routing information and then listens for responses to the request. If a system configured to supply RIP hears the request, it responds with a response packet based on information in its routing database. The response packet contains destination network addresses and the routing metric for each destination.

When a RIP response packet is received, the routing daemon takes the information and rebuilds the routing database adding new routes and "better" (lower metric) routes to destinations already listed in the database. RIP also deletes routes from the database if the next router to that destination indicates that the route contains more than 15 hops, or if the route is deleted. If no updates are received from that gateway for a specified time period, all routes through a gateway are deleted. Generally, routing updates are issued every 30 seconds. In many implementations, if a gateway is not heard from for 180 seconds, all routes from that gateway are deleted from the routing database. This 180-second interval also applies to deletion of specific routes.

RIP version 2 (more commonly known as RIP II) adds additional capabilities to RIP. For more information about RIP II, see RFC 1388.

The syntax for the `rip` statement follows:

```
rip yes | no | on | off [ {
 broadcast ;
 nobroadcast ;
 nocheckzero ;
 preference preference ;
 defaultmetric metric ;
 interface interface_list
  [noripin]
  [noripout]
  [metricin metric]
  [metricout metric]
  [version 1]|[version 2 [multicast| broadcast]]
  [authentication [none | password]] ;
 trustedgateways gateway_list ;
 sourcegateways gateway_list ;
 traceoptions trace_options ;
} ] ;
```

The `rip` statement enables or disables RIP. If you do not specify the `rip` statement, the default is `rip on`. If enabled, RIP assumes `nobroadcast` when only one interface exists and `broadcast` when more than one exists.

The options are as follows:

`broadcast` Specifies that RIP packets are broadcast regardless of the number of interfaces present. This option is useful when propagating static routes or routes learned from anther protocol into RIP. In some cases, the use of `broadcast` when only one network interface is present can cause data packets to traverse a single network twice.

`nobroadcast` Specifies that RIP packets are not broadcast on attached interfaces, even if more than one exists. If a `sourcegateways` clause is present, routes are still unicast directly to that gateway.

`nocheckzero` Specifies that RIP should not check to make sure that reserved fields in incoming version 1 RIP packets are 0. Usually, when the reserved fields are not 0, RIP rejects packets.

`preference` *preference*
: Sets the preference for routes learned from RIP. The default preference is 100. To override this preference, specify a preference in import policy.

`defaultmetric` *metric*
: Defines the metric used when advertising routes by using RIP that were learned from other protocols. If you omit this option, the default value is 16 (unreachable). This choice of default values requires you to specify a metric explicitly to export routes from other protocols into RIP. To override this metric, specify a metric in export policy.

`interface` *interface_list*
: Controls various attributes of sending RIP on specific interfaces. For the description of the *interface_list*, see the section on *interface_list* specification. The following are the possible parameters:

  `noripin` Specifies that RIP packets received using the specified interface are ignored. The default is to listen to RIP on all interfaces.

  `noripout` Specifies that no RIP packets are sent on the specified interfaces. When in `broadcast` mode, the default is to send RIP on all interfaces.

  `metricin` *metric*
  : Specifies the RIP metric to add to incoming routes before they are installed in the routing table. The default is the kernel interface metric plus 1 (which is the default RIP hop count). If you specify this value, it is used as the absolute value. The kernel metric is not added. This option is used to make RIP routes learned through the specified interfaces less preferable than RIP routes from other interfaces.

`metricout` *metric*
: Specifies the RIP metric to be added to routes that are sent using the specified interfaces. The default is 0. This option is used to make other routers prefer RIP routes from other interfaces over RIP routes learned using the specified interfaces.

`version 1` Specifies that RIP packets sent using the specified interfaces are version 1 packets. This is the default.

`version 2` Specifies that RIP version 2 packets are sent on the specified interfaces. If Internet Protocol (IP) multicast support is available on this interface, the default is to send full version 2 packets. If it is not available, version 2 packets that are compatible with version 1 are sent.

`multicast` Specifies that RIP version 2 packets should be multicast on this interface. This is the default for RIP version 2.

`broadcast` Specifies that RIP version 2 packets that are compatible with version 1 should be broadcast on this interface, even if IP multicast is available.

`authentication`
: Defines the authentication type to use. It applies only to RIP version 2 and is ignored for RIP version-1 packets. The default authentication type is `none`. If you specify a *password*, the authentication type used is simple. The *password* should be a quoted string between 0 and 16 characters.

`trustedgateways` *gateway_list*
: Defines the list of gateways from which RIP will accept updates. The *gateway_list* is simply a list of host names or IP addresses. By default, all routers on the shared network are trusted to supply routing information. But, if you specify the `trustedgateways` clause, only updates from the gateways in the *gateway_list* are accepted.

`sourcegateways` *gateway_list*
: Defines a list of routers to which RIP sends packets directly, not through multicast or broadcast. By default, RIP packets are broadcast to every system on the shared network. If you use the `sourcegateways` statement, updates are sent only to the gateways in the *gateway_list*.

`traceoptions` *trace_options*
: Specifies the tracing options for RIP (see the Trace Options subsection of this man page).

Open Shortest Path First (OSPF) protocol

Open Shortest Path First (OSPF) routing protocol is a shortest path first (SPF) or link-state protocol. OSPF is an interior gateway protocol that distributes routing information between routers in a single autonomous system. OSPF is suitable for complex networks that have many routers. Each network that has at least two attached routers has a designated router and a back-up designated router. The designated router floods a link-state advertisement for the network and has other special responsibilities. The designated router concept reduces the number of adjacencies required on a network.

OSPF allows networks to be grouped into areas. Routing information passed between areas is abstracted, potentially allowing a significant reduction in routing traffic. OSPF uses four different types of routes, listed in order of preference: intra-area, inter-area, type 1 external, and type 2 external. Intra-area paths have destinations within the same area; inter-area paths have destinations in other OSPF areas; and Autonomous System External (ASE) routes are routes to destinations external to the AS. Routes imported into OSPF as type 1 ASE routes are supposed to be from peers whose external metrics are directly comparable to OSPF metrics. Type 2 ASEs are used for peers whose metrics are not comparable to OSPF metrics.

OSPF intra- and inter-area routes are always imported into the `gated`(8) routing database with a preference of 10. If an OSPF router did not participate fully in the area's OSPF, it would be a violation of the protocol, it would update the protocal; therefore, you cannot override this. Although you can give other routes lower preference values explicitly, you should not do so.

Hardware multicast capabilities also are used when possible to deliver link-status messages.

OSPF areas are connected by the backbone area, the area with identifier 0.0.0.0. All areas must be logically contiguous, and the backbone is no exception. To permit maximum flexibility, OSPF allows the configuration of virtual links to enable the backbone area to appear contiguous despite the physical reality.

All routers in an area must agree on that area's parameters. Most configuration parameters are defined on a per area basis. All routers that belong to an area must agree on that area's configuration.

`ospf` Statement

The syntax for the `ospf` statement follows:

```
ospf yes | no | on | off [ {
  defaults {
      preference preference ;
      cost cost ;
      tag [ as ] tag ;
      type 1 | 2 ;
  } ;
  exportlimit routes ;
  exportinterval time ;
  traceoptions trace_options ;
  monitorauthkey authkey ;
  backbone | ( area area ) {
        authtype 0 | 1 | none | simple ;
        stub [ cost cost] ;
        networks {
              network ;
              network mask mask ;
              network masklen number ;
              host host ;
        } ;
        stubhosts {
              host cost cost ;
```

```
            } ;
            interface interface_list; [cost cost ] {
                  interface_parameters
            } ;
            interface interface_list nonbroadcast [cost cost ] {
                  pollinterval time ;
                  routers {
                        gateway [ eligible ] ;
                  } ;
                  interface_parameters
            } ;
            Backbone only:
            virtuallink neighborid router_id transitarea area {
                  interface_parameters
        } ;
      } ;
    } ] ;
```

The following are the *interface_parameters* referred to previously. You may specify them on any class of interface, and they are described under the `interface` clause.

```
    enable | disable ;
    retransmitinterval time ;
    transitdelay time ;
    priority priority ;
    hellointerval time ;
    routerdeadinterval time ;
    authkey auth_key ;
```

defaults

These parameters specify the `defaults` used when importing OSPF ASE routes into the `gated`(8) routing table and exporting routes from the `gated`(8) routing table into OSPF ASEs.

`preference` *preference*
: The *preference* determines how OSPF routes compete with routes from other protocols in the `gated` routing table. The default value is 150.

`cost` *cost*
: The cost is used when exporting a non-OSPF route from the `gated` routing table into OSPF as an ASE. The default value is 1. You may explicitly override this value in the export policy.

`tag` [`as`] *tag*

OSPF ASE routes have a 32-bit tag field that the OSPF protocol does not use, but which export policy may use to filter routes. When OSPF is interacting with an exterior gateway protocol, the tag field may be used to propagate AS path information; in which case, the `as` keyword is specified because the tag is limited to 12 bits of information. If you omit this parameter, the tag is set to 0.

`type 1 | 2` Routes exported from the `gated` routing table into OSPF default to becoming type 1 ASEs. You may explicitly change this default here and override it in the export policy.

Because of the nature of OSPF, you must limit the rate at which ASEs are flooded. To adjust those rate limits, use the following two parameters:

`exportinterval` *time*

Specifies how often a batch of ASE link-state advertisements are generated and flooded into OSPF. The default is once per second.

`exportlimit` *routes*

Specifies how many ASEs are generated and flooded in each batch. The default is 100.

`traceoptions` *trace_options*

Specifies the tracing options for OSPF. See the Trace Options subsection and the OSPF-specific tracing options.

`monitorauthkey` *authkey*

OSPF state may be queried using the `ospf_monitor` utility. This utility sends nonstandard OSPF packets, which generate a text response from `gated`(8). By default, these requests are not authenticated. If an authentication key is configured, the incoming requests must match the specified authentication key. These packets cannot change an OSPF state, but the act of querying OSPF can use system resources.

`backbone area |` *area*

You must configure each OSPF router into at least one OSPF area. If you configure more than one area, at least one must be the backbone. You can configure the backbone only by using the `backbone` keyword; you cannot specify it as `area 0`. Each area must have at least one interface. The backbone interface may be a `virtuallink`.

`authtype 0 | 1 | none | simple`

OSPF specifies an authentication scheme per area. Each interface in the area must use this same authentication scheme although it may use a different `authenticationkey`. The currently valid values are `none` (`0`) for no authentication, or `simple` (`1`) for simple password authentication.

`stub` [`cost` *cost*]

A `stub` area is one in which there are no ASE routes. If you specify a `cost`, this is used to inject a default route into the area with the specified cost.

`networks` The `networks` list describes the scope of an area. Intra-area LSAs that fall within the specified ranges are not advertised into other areas as inter-area routes. Instead, the specified ranges are advertised as `summary network` LSAs. Inter-area LSAs that do not fall into any range also are advertised as summary network LSAs. This option is very useful on well-designed networks in reducing the amount of routing information propagated between areas.

`stubhosts` This option specifies directly attached hosts that should be advertised as reachable from this router and the costs with which they should be advertised. You should specify point-to-point interfaces on which it is not desirable to run OSPF.

It also is useful to assign an additional address to the loopback interface (one not on the 127 network) and advertise it as a stub host. If this address is the same one used as the router ID, it enables routing to OSPF routers by router ID, rather than by interface address. This is more reliable than routing to one of the routers' interface addresses, which may not always be reachable.

`interface` *interface_list* [`cost` *cost*]

This form of the interface clause is used to configure a `broadcast` (which requires IP multicast support) or a `point-to-point` interface. For the description of *interface_list*, see the section on *interface_list* specification.

Each interface has a `cost`. The costs of all interfaces a packet must cross to reach a destination are summed to get the cost to that destination. The default `cost` is 1, but you may specify another nonzero value.

Interface parameters are common to all types of interfaces:

`retransmitinterval` *time*

The number of seconds between link-state advertisement retransmissions for adjacencies that belong to this interface.

`transitdelay` *time*

The estimated number of seconds required to transmit a link-state update over this interface. `transitdelay` takes into account transmission and propagation delays, and it must be greater than 0.

`priority` *priority*

A number between 0 and 255 that specifies the priority for becoming the designated router on this interface. When two routers attached to a network both try to become designated router, the one that has the highest priority wins. A router that has the router priority set to 0 is ineligible to become designated router.

`hellointerval` *time*

The length of time, in seconds, between Hello packets that the router sends on the interface.

`routerdeadinterval` *time*

If a neighbor router's Hello packet is not heard for *time* seconds, `gated` declares that the neighbor is down.

`authkey` *auth_key*

Used by OSPF authentication to generate and verify the authentication field in the OSPF header. You can configure the authentication key on a per interface basis. It is specified by 1 to 8 decimal digits separated by periods, a 1-to-8 byte hexadecimal string preceded by`0x`, or a 1-to-8 character string in double quotation marks.

`interface` *interface_list* `nonbroadcast` [`cost` *cost*]

This form of the interface clause is used to specify a `nonbroadcast` interface on a nonbroadcast multiaccess (NBMA) media. Because an OSPF `broadcast` media must support IP multicasting, you must configure a broadcast-capable media (such as Ethernet) that does not support IP multicasting as a nonbroadcast interface.

A nonbroadcast interface supports any of the preceding standard `interface` clauses, plus the following two that are specific to nonbroadcast interfaces:

`pollinterval` *time*

Before an adjacency is established with a neighbor, OSPF packets are sent periodically at the specified `pollinterval`.

`routers` By definition, you cannot send broadcast packets to discover OSPF neighbors on a nonbroadcast interface; therefore, you must configure all neighbors. The router list includes one or more neighbors and an indication of their eligibility to become a designated router.

`virtuallink neighborid` *router_id* `transitarea` *area*

Virtual links are used to establish or increase connectivity of the backbone area. The `neighborid` is the *router_id* of the other end of the virtual link. The transit *area* specified also must be configured on this system. You may specify all standard interface parameters defined by the `interface` clause on a virtual link.

Tracing options

In addition to the following OSPF-specific trace flags, OSPF supports the `all`, `ospf`, and `protocol` flags. The `protocol` flag traces interface and neighbor state machine transitions.

OSPF-specific trace flags are as follows:

`lsabuild`	Link State Advertisement creation
`spf`	Shortest Path First (SPF) calculations
`lsatransmit`	Link State Advertisement (LSA) transmission
`lsareceive`	LSA reception

You may modify the following packet tracing options by using the `update` flag:

`hello` OSPF `HELLO` packets, which are used to determine neighbor reachability.

`dd` OSPF Database Description packets, which are used to synchronize OSPF databases.

`request` OSPF Link State Request packets, which are used to synchronize OSPF databases.

`lsu` OSPF Link State Update packets, which are used to synchronize OSPF databases.

`ack` OSPF Link State Acknowledgment packets, which are used to synchronize OSPF databases.

`redirect` Statement

The syntax for the `redirect` statement follows:

```
redirect yes|no|on|off [ {
      preference preference ;
      interface interface_list ;
      trustedgateways gateway_list ;
} ] ;
```

Controls whether ICMP redirects are listened to. If you omit this statement, the default is to listen to ICMP redirects, unless RIP is enabled and more than one interface exists. When ICMP redirects are disabled, `gated` must actively remove the effects of redirects from the kernel, because the kernel always processes ICMP redirects.

The default preference is 30.

Route Statements

The `static` statements defines the static routes that `gated`(8) uses. A single `static` statement can specify any number of routes. The `static` statements occur after protocol statements and before control statements in the `gated.conf` file. You may specify any number of `static` statements, each containing any number of static route definitions. Routes from other protocols that have better preference values can override these routes.

```
static {
 (host host) | default | (network [ (mask mask) | (masklen number ) ] )
  gateway gateway_list
  [interface interface_list]
  [preference preference]
  [retain]
  [noinstall]
  [static_options] ;
 network [ (mask mask) | (masklen number) ] interface interface
  [preference preference]
  [retain]
  [noinstall]
  [static_options] ;
} ;
```

`host` *host* `gateway` *gateway_list*
network [(`mask` *mask*) | (`masklen` *number*)] `gateway` *gateway_list*
`default` `gateway` *gateway_list*

This is the most general form of the static statement. It defines a static route through one or more gateways. Static routes are installed when one or more of the `gateways` listed are available on directly attached interfaces. If more than one eligible gateway is available, they are limited by the number of multipath destinations supported (this compile time parameter is currently four).

Parameters for static routes are as follows:

`interface` *interface_list*
When you specify this parameter, gateways are considered valid only when they are on one of these interfaces. For the description of the *interface_list*, see the section on *interface_list* specification.

`preference` *preference*
This option selects the preference of this static route. The preference controls how this route competes with routes from other protocols. The default preference is 60.

`retain` Usually, `gated`(8) removes all routes except interface routes from the kernel forwarding table during a graceful shutdown. You may use the `retain` option to prevent specific static routes from being removed. This is useful to ensure that some routing is available when `gated`(8) is not running.

`noinstall` Typically, the route with the lowest preference is installed in the kernel forwarding table and is the route exported to other protocols. When you specify `noinstall` on a route, it is not eligible to be installed in the kernel forwarding table when it is active, but it is still eligible to be exported to other protocols.

network [(`mask` *mask*) | (`masklen` *number*)] `interface` *interface*
This form defines a static interface route that is used for primitive support of multiple network addresses on one interface. The `preference`, `retain`, and `noinstall` options are the same as described previously.

Static options are as follows:

`admmtu` *number*
Sets the maximum transmission unit (mtu) size for the route to *number*.

`genmask` *mask*
Sets the generation mask for the route to *mask*.

`gid` *grouplist*
Restricts the route that the groups specified in *grouplist* can use.

`netmask` *mask*
Sets the netmask for the route to *mask*. This is an obsolete form of specifying the netmask.

`service` *tos*
Specifies *tos* as the IP type of service.

`tosmatch` Indicates that a client must explicitly request (that is, match) the type of service specified for the route to be able to use it.

The following static options have no effect on the route table. They may be implemented in a future release of the UNICOS operating system.

`hopcount` *number*
Sets the hopcount (number of gateway hops to the destination of) for the route to *number*.

`expire` *number*
Sets the lifetime (in seconds) for the route to *number*.

`lock` Indicates that the next option must be locked against further changes.

`lockrest` Indicates that all remaining options specified for the route must be locked against further changes.

`mtu` *number* Sets the maximum transmission unit size for the route to *number*.

`recvpipe` *number*
Sets the inbound delay-bandwidth product for the route to *number*.

`rtt` *number* Sets the estimated round-trip time for the route to *number*.

`rttvar` *number*
Sets the estimated round-trip time variance for the route to *number*.

`sendpipe` *number*
Sets the outbound delay-bandwidth product for the route to *number*.

`ssthresh` *number*
Sets the outbound gateway buffer limit for the route to *number*.

Control Statements

The `import` and `export` statements control importation of routes from routing protocol peers and exportation of routes to routing protocol peers. In the following `import` and `export` statement formats, the use of the token `all` for *interface_list* is redundant; therefore, it is not allowed.

The `import` statement can have the following syntax:

```
import proto rip|redirect restrict ;

import proto rip|redirect
   [preference preference] {
   import_list
} ;

import proto rip|redirect interface interface_list restrict ;

import proto rip|redirect interface interface_list
   [preference preference] {
   import_list
} ;

import proto rip|redirect gateway gateway_list restrict ;

import proto rip|redirect gateway gateway_list
   [preference preference] {
   import_list
} ;

import proto ospfase [tag ospf_tag] restrict ;

import proto ospfase [tag ospf_tag]
   [preference preference] [{
   import_list
}] ;
```

If you specify an *ospf_tag* specification, only routes matching that tag specification are considered; otherwise, any tag is considered. An OSPF tag specification may be a decimal, hexadecimal, or a dotted quad number.

If you specify more than one import statement relevant to a protocol, they are processed most specific to least specific (for example, for RIP, gateway, interface, and protocol), then in the order specified in the configuration file.

The import statement `restrict` parameter causes routes learned by the import statement to be ignored. The `preference` parameter specifies the preference of routes learned from this import statement.

The following is the format of an *import_list*:

```
dest_mask [[restrict] | [preference preference]] ;
```

An *import_list* consists of zero or more destinations (with optional mask). You may specify one of two parameters: `restrict` to prevent a set of destinations from being imported, or a specific `preference` for this set of destinations.

The contents of an *import_list* are sorted internally so that entries that have the most specific masks are examined first. The order in which *dest_mask* entries are specified does not matter.

If you specify an import list, the import list is scanned for a match. If no match is found, the route is discarded. An `all restrict` entry is assumed in an import list.

The `export` statement can have the following formats:

```
export proto rip restrict ;

export proto rip [metric metric] {
   export_list
} ;

export proto rip interface interface_list restrict ;

export proto rip interface interface_list
   [metric metric] {
   export_list
} ;

export proto rip gateway gateway_list restrict ;

export proto rip gateway gateway_list
   [metric metric] {
   export_list
} ;

export proto ospfase [type 1|2] [tag ospf_tag] restrict ;

export proto ospfase [type 1|2] [tag ospf_tag]
   [cost ospf_cost] {
   export_list
} ;
```

The export statement distributes routes to a destination protocol, gateway, or interface. The `restrict` parameter prevents the routes specified by the export statement from being exported.

The export list specifies the source of the routes that are distributed by the export statement. The format of an export list follows:

```
proto rip|direct|static
   [restrict] | [metric metric] [ {
   announce_list
} ] ;

proto rip|direct|static interface interface_list
   [restrict] | [metric metric] [ {
   announce_list
} ] ;

proto rip gateway gateway_list
   [restrict] | [metric metric] [ {
   announce_list
} ] ;

proto ospf [restrict] | [metric metric] [ {
   announce_list ;
} ] ;

proto ospfase [restrict | metric metric]] [ {
   announce_list ;
} ] ;

proto proto tag tag
   [restrict] | [metric metric] [ {
   announce_list
} ] ;
```

If you specify a tag, only routes with that tag will be considered; otherwise, any tag will be considered. An OSPF tag on an export statement may be a decimal or hexadecimal. An OSPF tag on an export list is a 31-bit number that is matched against the tag present (if any) on that route.

If you specify more than one export statement relevant to a protocol, they are processed most specific to least specific (for example, for RIP, gateway, interface, and protocol), then in the order specified in the configuration file.

By default, interface routes are exported to all protocols. RIP also exports its own routes. An export specification that has only a `restrict` prevents these defaults from being exported. You cannot change the metric RIP uses for its own routes; if you try to override this metric, it is silently ignored.

You may specify any protocol for import lists that referr to AS paths and tags. Tags are currently meaningful only for OSPF ASE routes.

An *announce_list* consists of zero or more destinations (with optional mask). You may specify one of two parameters: `restrict` to prevent a set of destinations from being exported, or a specific metric for this set of destinations.

dest_mask [[`restrict`] | [`metric` *metric*]] ;
. . .

The contents of an *announce_list* are sorted internally so that entries that have the most specific masks are examined first. The order in which *dest_mask* entries are specified does not matter.

If you omit *announce_list*, all destinations are announced. If you specify an announce list, an `all restrict` is assumed. Therefore, an empty announce list is the equivalent of `all restrict`.

To announce routes that specify a next hop of the loopback interface (for example, static routes) through RIP, you must specify the metric at some level in the export clause; setting a default metric for RIP is not sufficient.

FILES

`/etc/gated.conf`

SEE ALSO

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

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

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

Routing Information Protocol, RFC 1058

Routing Information Protocol version 2, RFC 1388

Open Shortest Path First Protocol version 2, RFC 1583

COPYRIGHT INFORMATION

NAME

gettydefs – Speed and terminal settings used by getty

IMPLEMENTATION

CRAY Y-MP systems

DESCRIPTION

The /etc/gettydefs file contains information that getty(8) uses to set up the speed and terminal settings for a line. This method of terminal handling is used for IOS terminals.

The information in the gettydefs file also specifies the appearance of the login prompt (usually login: by default).

Each entry in the /etc/gettydefs file has the following format:

label# *initial-flags* # *final-flags* # *login-prompt* #*next-label*

Each entry is followed by a blank line. The various fields can contain quoted characters of the form \b, \n, or \c, as well as *nnn*; *nnn* is the octal value of the desired character. The various fields are as follows:

label String against which getty(8) tries to match its second argument. This is often the baud rate at which the terminal is supposed to run (for example, 1200) but it need not be (see the definition of *next-label*). Speed settings have no effect on IOS terminals.

initial-flags Initial ioctl(2) settings to which the terminal will be set if a terminal type is not specified to getty(8). These flags are the same as those in the sys/termio.h include file. Usually, only the speed flag is required in *initial-flags*. The getty(8) program automatically sets the terminal to raw input mode and handles most of the other flags. The *initial-flag* settings remain in effect until getty(8) executes login(1).

final-flags These flags accept the same values as the *initial-flags* and are set just before getty(8) executes login(1). The speed flag is again required. The SANE composite flag handles most of the other flags that must be set so that the processor and terminal are communicating according to the same protocol. Two commonly specified *final-flags* are TAB3, which send tabs to the terminal as spaces, and HUPCL, which hangs up the line on the final close.

login-prompt This entire field is printed as the login prompt. Unlike the preceding fields, in which white space (a space, tab, or newline character) is ignored, it is included in the login prompt field.

next-label If this entry does not specify the desired speed, indicated by the typing of a break character, getty(8) searches for the entry with *next-label* as its *label* field and sets up the terminal for those settings. Usually, a series of speeds is linked in this fashion to form a closed set; for example, 2400 is linked to 1200, which in turn is linked to 300, which finally is linked to 2400.

If getty(8) is called without a second argument, the first entry of gettydefs is used, thus making the first entry of gettydefs the default entry. It also is used if getty(8) cannot find the specified *label*. If the gettydefs file itself is missing, one entry is built into getty(8) that will bring up a terminal (at 9600 Bd).

After you create or modify a gettydefs file, run it through getty(8) by using the -c (check) option to ensure that no errors exist.

FILES

/etc/gettydefs	File of terminal information used by getty(8)
/usr/include/sys/termio.h	Structure used by ioctl(2) system calls to terminal devices

SEE ALSO

termio(4)

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

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

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

NAME

group – Format of the group-information file

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The /etc/group file contains the following information for each user group:

- Group name
- Encrypted password
- Numeric group ID (GID)
- Comma-separated list of user names allowed in the group

The group file is an ASCII file. The fields are separated by colons; each group is separated from the next by a newline character. udbgen(8) automatically maintains this file to match the information in the udb(5) file. The password field is present for compatibility, but it is always set to *.

This file resides in the /etc directory and has general read permission so that it can be used, for example, to map numeric group IDs to names.

NOTES

The encrypted password field is available under the UNICOS operating system, but Cray Research does not support it.

The list of user names can become very long for groups shared by many users. To keep the line length reasonable, udbgen(8) generates the group file that has a maximum membership list of about 400 characters. If the group list exceeds this length, additional lines are created to hold the remainder of the list. The additional lines will be adjacent and will begin with the identical group name and group ID. For example, if the group list for group gr1 with GID 123 were long enough to occupy three lines, that fragment of the group file would appear as follows:

```
gr1:*:123:usr1,usr2,usr3,usr4,usr6,usr10
gr1:*:123:usr101,usr102,usr103,usr104,usr105
gr1:*:123:usr563,usr570
```

The first two lines would have a group list that consists of about 400 characters (the example shows a short list for brevity) and the final line would consist of the remainder of the list. The 400-character limit is approximate because the line is broken at the end of the name that causes the length to exceed 400 characters.

Unlike the /etc/passwd file, you must update the /etc/group file manually to include new group IDs and group names. When you update /etc/group, ensure that the udbgen(8) utility is not running, because udbgen would overwrite any changes to /etc/group.

FILES

`/etc/group` File that contains user group information

SEE ALSO

`acid`(5), `passwd`(5), `udb`(5)

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

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

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

NAME

hosts, hosts.bin – Contains network host name database

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The /etc/hosts file contains the database of all locally known hosts on the TCP/IP network. The /etc/hosts.bin file is the binary version of the hosts file; the mkbinhost(8) command creates it.

For each host, one line should contain the network type (optional), the host's Internet address, the official host name, and any aliases that exist for the host name. The recognized value for the network type field is inet (the default). Items are separated by any number of blanks and/or tab characters. A # symbol indicates the beginning of a comment; when you use the # symbol, the routines that search the file ignore additional characters up to the end of the line. The hosts file is searched sequentially; therefore, if you specify more than one host name with a given Internet address, the first entry is used and all others are ignored. Specify Internet network addresses in the conventional "." (dot) notation, using the inet_addr routine from the Internet address manipulation library, inet(3C).

Host names can contain any printable character other than a blank, tab, new line, or comment (#).

NOTES

All library routines in gethost(3C) check for the existence of the /etc/hosts.usenamed file. If it exists, they use the domain name service (see resolver(3C)) to perform host name and address lookups; otherwise, they use hosts.bin if it exists. If it does not exist, the library routines get information from hosts. When /etc/hosts is modified, you should run mkbinhost to update /etc/hosts.bin.

Avoid using both uppercase and lowercase letters in hosts names, because some implementations of TCP/IP cannot handle mixed-case host names.

EXAMPLES

The following is an example of entries in an `/etc/hosts` file:

```
#
#     HYPERchannel addresses
#
84.0.0xc4.5 sn101      sn101-inet
84.0.0x13.0 nobel      nobel-inet
#
#     Ethernet addresses
#
#192.9.1    nobelnet
192.9.1.17  nobel mailhost
192.9.1.18  ranger
192.9.1.19  lps
192.9.1.20  sol
```

FILES

`/etc/hosts`	File that contains names of known hosts on TCP/IP network.
`/etc/hosts.bin`	Binary version of `/etc/hosts` file.
`/etc/hosts.usenamed`	The existence of this file turns on domain name service (`named`) lookup for host names and addresses.

SEE ALSO

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

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

NAME

`hosts.equiv` – Contains public information for validating remote autologin

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `/etc/hosts.equiv` and `.rhosts` files provide the remote authentication database for `rlogin`(1B), `remsh`(1B), `rcp`(1), and `rcmd`(3C). If the Network Queuing System (NQS) is using file validation, it also will use the `/etc/hosts.equiv` file. If a `.nqshosts` file does not exist, NQS will use the `.rhosts` file. The files specify remote hosts and users that are considered trusted. Trusted users are allowed to access the local system without supplying a password. The `ruserok`() library routine (see `rcmd`(3C)) performs the authentication procedure for programs by using the `/etc/hosts.equiv` and `.rhosts` files. The `/etc/hosts.equiv` file applies to the entire system, but individual users can maintain their own `.rhosts` files in their home directories.

These files bypass the standard password-based user authentication mechanism. To maintain system security, you must take care when creating and maintaining these files.

The remote authentication procedure determines whether a remote user from a remote host should be allowed to access the local system as a (possibly different) local user. This procedure first checks the `/etc/hosts.equiv` file and then checks the `.rhosts` file in the home directory of the local user for whom access is being tried. Entries in these files can be positive entries, which explicitly allow access, and negative entries, which explicitly deny access. The authentication succeeds as soon as a matching positive entry is found. The procedure fails when a matching negative entry is found or if no matching entry is found in either file. The order of entries, therefore, can be important; if the file contains both matching positive and negative entries, the entry that appears first will prevail. If the remote authentication procedure fails, the `remsh`(1B) and `rcp`(1) programs fail, but the `rlogin`(1B) command falls back to the standard password-based login procedure.

Both the `/etc/hosts.equiv` and `.rhosts` files are formatted as a list of one-line entries. Each entry has the following form:

> *hostname* [*username*]

If the following form is used, users from the host *hostname* are trusted; that is, they may access the system by using the same user name as they have on the remote system.

> *hostname*

You may use this form in both the `/etc/hosts.equiv` and `.rhosts` files.

If the line is in the following form, the user *username* from the host *hostname* can access the system:

> *hostname username*

You may use this form in individual `.rhosts` files to allow remote users to access the system as a different local user. If this form is used in the `/etc/hosts.equiv file`, the user *username* is allowed to access the system as any local user.

Negative entries disallow access and are preceded by a – symbol. The following form disallows access by the user *username* only from the host *hostname*:

hostname – *username*

The following form disallows all access from the host *hostname*:

– *hostname*

To match all users or all hosts, use an `*` symbol. For example, entering `*` allows any user from any host to log in under the same user name. Entering `*` *username* allows the user *username* access from any remote host. Entering *hostname* `*` in a `.rhosts` file allows any user from the remote host *hostname* to access the system as the user in whose `.rhosts` file the entry appeared.

You should use positive entries in `/etc/hosts.equiv` that include a *username* field with extreme caution. Because `/etc/hosts.equiv` applies systemwide, these entries can allow one, or a group of, remote users access to the system as any local user. This can be a security problem.

NOTES

To authenticate the user, the system configuration can require an entry in both the user's `.rhost` and `/etc/hosts.equiv` files, and it also may require the remote user name to match the local user name.

FILES

`/etc/hosts.equiv`	File that contains name(s) for a remote host
`/etc/udb`	File that contains remote user names

SEE ALSO

`rhosts`(5)

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

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

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

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

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

UNICOS NQS and NQE Administrator's Guide, Cray Research publication SG–2305

NAME

`inetd.conf` – Internet super-server configuration file

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `/etc/inetd.conf` file contains the configuration information used by the Internet super-server configuration file, which listens for incoming service requests. The `inetd`(8) command is invoked at boot time.

Upon execution, `inetd` reads its configuration information from a configuration file which, by default, is `/etc/inted.conf`. There must be an entry for each field of the configuration file, with entries for each field separated by a tab or a space. Comments are denoted by a "#" at the beginning of a line.

The fields of the configuration file are as follows:

service name There are several types of services that `inetd` can start: `standard`, `TCPMUX`, and `RPC`. The *service name* field consists of a valid service in the `/etc/services` file or a port number on which the `inetd` daemon can listen for incoming requests. For internal services, the server name must be the official name of the service (the first entry in `/etc/services`). For `TCPMUX` services, the value of the *service name* field consists of the string *tcpmux* followed by a slash and the locally-chosen service name. The service names listed in `/etc/services` and the name *help* are reserved. Try to choose unique names for your `TCPMUX` services by prefixing them with your organization's name and suffixing them with a version number. For `RPC` services, *service name* consists of the `RPC` *service name* followed by a slash and either a version number or a range of version numbers (e.g., *rstat/2-4*).

socket type Should be one of the following values:

`dgram`	Indicates that the socket is a datagram
`raw`	Indicates that the socket is raw
`rdm`	(Not implemented) Indicates that the socket is a reliably delivered message
`seqpacket`	(Not implemented) Indicates that the socket is a sequenced packet socket
`stream`	Indicates that the socket is a stream. `TCPMUX` services must use `stream`.

protocol The valid protocol as given in the `/etc/protocols` file. Protocol is usually `tcp` or `udp`. `TCPMUX` services must use *tcp*. For RPC services, *protocol* consists of the string `rpc` followed by a slash and the name of the protocol. For example, `rpc/tcp` indicates that an RPC server is using the TCP protocol as its transport mechanism.

`wait|nowait` This 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 and thus frees the socket so that `inetd`(8) can receive further messages on the socket, it is a multithreaded server, and it should use 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. `talk` is an example of the latter type of datagram server.

The `tftpd` server is an exception; it is a datagram server that establishes pseudo-connections. To avoid a race, it must be listed as `wait`; 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.

`TCPMUX` services must use `nowait`.

user User name of the user as whom the server should run. By associating a user with the daemon, servers can be given less permission than root. The `ftp`, `telnet`, `shell`, and `login` servers need root permission; the `finger` and `tftp` servers should be run as a user with limited capability.

server program Path name of the program that `inetd` will execute when a request is found on its socket. If `inetd` provides this service internally, this entry should be `internal`.

server program arguments

Arguments to the `exec`(2) system call, starting with *argv*[0], which is the name of the program.

CAUTIONS

For security reasons, you should use `fingerd`(8) as a user with limited priority and disable `tftpd`(8).

TCPMUX

RFC 1078 describes the `TCPMUX` protocol: "A `TCP` client connects to a foreign host on TCP port 1. It sends the service name followed by a carriage-return line-feed <CRLF>. The service name is never case sensitive. The server replies with a single character indicating positive (+) or negative(-) acknowledgement, immediately followed by an optional message of explanation, terminated with a <CRLF>. If the reply was positive, the selected protocol begins; otherwise the connection is closed." The program is passed the `TCP` connection as file descriptors 0 and 1.

If the `TCPMUX` service name begins with a "+", `inetd` returns the positive reply for the program. This allows you to invoke programs that use `stdin/stdout` without putting any special server code in them.

The special service name *help* causes `inetd` to list `TCPMUX` services in `inetd.conf`.

EXAMPLES

Following is a sample `inetd.conf` file. In this example, the services `uucp`, `tftp`, `comsat`, `talk`, and `ntalk` are not needed and have been commented out. Each service is listed with its associated socket type and protocol; use this example as a reference to socket types and protocols.

```
#
# Internet server configuration database
#
ftp       stream  tcp  nowait  root    /etc/ftpd  ftpd
telnet    stream  tcp  nowait  root    /etc/telnetd  telnetd
shell     stream  tcp  nowait  root    /etc/rshd  rshd
login     stream  tcp  nowait  root    /etc/rlogind  rlogind
exec      stream  tcp  nowait  root    /etc/rexecd  rexecd
# Run as user "uucp" if you don’t want uucpd’s wtmp entries.
#uucp     stream  tcp  nowait  root    /etc/uucpd  uucpd
finger    stream  tcp  nowait  nobody  /etc/fingerd  fingerd
#tftp     dgram   udp  wait    tftp    /etc/tftpd  tftpd
#comsat   dgram   udp  wait    root    /etc/comsat  comsat
#talk     dgram   udp  wait    root    /etc/talkd  talkd
#ntalk    dgram   udp  wait    root    /etc/ntalkd  ntalkd
echo      stream  tcp  nowait  root    internal
discard   stream  tcp  nowait  root    internal
chargen   stream  tcp  nowait  root    internal
daytime   stream  tcp  nowait  root    internal
time      stream  tcp  nowait  root    internal
tcpmux    stream  tcp  nowait  root    internal
echo      dgram   udp  wait    root    internal
discard   dgram   udp  wait    root    internal
chargen   dgram   udp  wait    root    internal
daytime   dgram   udp  wait    root    internal
time      dgram   udp  wait    root    internal
#
# TCPMUX service syntax:
#
tcpmux/+date  stream tcp nowait guest /bin/date date
tcpmux/phonebook stream tcp nowait guest /usr/local/bin/phonebook phonebook
#
#
# RPC services syntax:
# <rpc_prog>/<vers> <socket_type> rpc/<proto> <flags> <user> <pathname> <args>
ypupdated/1   stream  rpc/tcp wait root /etc/ypupdated ypupdated
rstatd/2-4    dgram   rpc/udp wait root /etc/rstatd    rstatd
rusersd/1-2   dgram   rpc/udp wait root /etc/rusersd   rusersd
sprayd/1      dgram   rpc/udp wait root /etc/sprayd    sprayd
rwalld/1      dgram   rpc/udp wait root /etc/walld     rwalld
```

FILES

`/etc/inetd.conf`	Contains configuration information used by the Internet super-server configuration file
`/etc/protocols`	Lists the valid protocols
`/etc/services`	Lists valid services

SEE ALSO

`protocols`(5), `services`(5)

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

RFC 1078: *TCP Port Service Multiplexer (TCPMUX)*

NAME

infoblk – Loader information table

SYNOPSIS

```
#include <infoblk.h>
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The segldr(1) and ld(1) commands build the infoblk loader information table into all normal executable programs. The contents of _infoblk provide information about the time and date of program creation, size and structure of the program's memory usage, and so on. To access the table contents within the program, use the C global structure name _infoblk, as follows:

```
extern struct infoblk _infoblk;

/*
 *  the infoblk structure, which SEGLDR puts in every binary
 *  (referenced by _infoblk)
 */

struct infoblk {
  unsigned i_vers: 7;       /* version of _infoblk table           */
  unsigned : 24;            /* unused                              */
  unsigned i_a: 1;          /* fill address generation flag        */
  unsigned i_len: 32;       /* no. of words in infoblk             */

  char i_name[8];           /* table name  - "infoblk"             */
  long i_cksum;             /* table checksum                      */
  char i_date[8];           /* date of program creation            */
  char i_time[8];           /* time of program creation            */
  char i_pid[8];            /* name of generating program          */
  char i_pvr[8];            /* version of generating program       */
  char i_osvr[8];           /* O.S. version at generation time     */
  long i_udt;               /* creation timestamp                  */
  long i_fill;              /* value to fill uninitialized areas   */

  unsigned i_tbase: 32;     /* text area base address              */
  unsigned i_dbase: 32;     /* data area base address              */

  unsigned i_tlen: 32;      /* text section length                 */
  unsigned i_dlen: 32;      /* data section length                 */
```

```
        unsigned i_blen: 32;      /* bss section length                     */
        unsigned i_zlen: 32;      /* zeroset section length                 */

        unsigned i_cdatalen: 32;  /* data section length before expansion */
        unsigned i_lmlen: 32;     /* CRAY-2 local memory length             */

        unsigned i_amlen: 32;     /* auxiliary memory length                */
        unsigned i_mbase: 32;     /* base address of heap                   */

        unsigned i_hinit: 32;     /* initial heap size                      */
        unsigned i_hinc: 32;      /* heap increment                         */

        unsigned i_sinit: 32;     /* initial stack size                     */
        unsigned i_sinc: 32;      /* stack increment                        */

        unsigned i_usxf: 32;      /* USX table first address                */
        unsigned i_usxl: 32;      /* USX table last address                 */

        unsigned i_mtptr: 32;     /* machine targeting block address        */
        unsigned i_cmptr: 32;     /* expansion entry list address           */

        unsigned i_enlen: 32;     /* saved arg/env length                   */
        unsigned i_enptr: 32;     /* environment pointer (currently 0)      */

        unsigned i_sgptr: 32;     /* pointer to $SEGRES table               */
        unsigned : 32;            /* unused                                 */

        unsigned i_taskstk: 32;   /* slave task initial stack size          */
        unsigned i_taskincr: 32;  /* slave task increment value             */

        long i_user1;             /* user value word 1                      */
        long i_user2;             /* user value word 2                      */
      };
```

FILES

`/usr/include/infoblk.h` Loader information table include file

SEE ALSO

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

NAME

`inittab` – Script for `init`(8) process

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `/etc/inittab` file is the script for `init`(8), the general process spawner. Processes dispatched by `init`(8) are, typically, the daemons required for the multiuser run levels and the shell for the UNICOS system console.

The `inittab` file is composed of position-dependent entries that have the following format:

id : *rstate* : *action* : *process*

A newline character delimits each entry; however, a backslash (`\`) preceding a newline character indicates a continuation of the entry. Up to 512 characters for each entry are permitted. You may insert comments in the *process* field, using the `sh`(1) convention for comments. No limits (other than the maximum entry size) are imposed on the number of entries in the `inittab` file. The entry fields are as follows:

id One to four characters used to identify an entry uniquely.

rstate Run level in which this entry will be processed. A run level is a configuration of the system; each run level allows only a selected group of processes to exist. Each process that `init`(8) spawns is assigned one or more run levels in which it is allowed to exist. Multiuser mode consists of seven run levels. The run levels are represented by a number that ranges from 0 through 6. For example, if the system is in run-level 1, only entries that have a 1 in the *rstate* field are processed.

When `init`(8) is requested to change run levels, all processes that do not have an entry in the *rstate* field for the target run level are sent a warning signal (`SIGTERM`) and allowed 20 seconds before being forcibly terminated by a kill signal (`SIGKILL`).

The *rstate* field can define multiple run levels for a process by selecting more than one run level in any combination from 0 through 6. If you do not specify a run level, *action* is taken on this process for all run levels 0 through 6.

Three other values (`a`, `b`, and `c`) can appear in the *rstate* field, even though they are not true run levels. Entries that have these values in the *rstate* field are processed only when the `telinit`(8) process (see `init`(8)) requests that they be run (regardless of the current run level of the system). They differ from run levels in that the system is in these states only for as long as it takes to execute all entries associated with the states. A process that an `a`, `b`, or `c` command starts is not killed when `init`(8) changes levels. It is killed only if its line in `/etc/inittab` is marked `off` in the *action* field, its line is deleted entirely from `/etc/inittab`, or `init`(8) goes into the single-user state.

action Keywords in this field specify how to treat the process in the *process* field. `init`(8) recognizes the following actions:

`boot` The `init`(8) command processes the entry only when reading the `inittab` file at boot time; `init`(8) starts the process and does not wait for its termination. When the process dies, `init`(8) does not restart it. For this instruction to be meaningful, *rstate* should be either the default or a match of the run level of `init`(8) at boot time. This action is useful for an initialization function that follows a hardware reboot of the system.

`bootwait` The `init`(8) command processes the entry only when reading the `inittab` file at boot time; `init`(8) starts the process and waits for its termination. When the process dies, `init`(8) does not restart it.

`generic` When a privileged daemon process initiates a new login session, it sends a request to `init`(8) through the `/etc/initreq` pipe (FIFO special file). This request includes the terminal to be used, the associated remote host, and the generic ID specified in the *id* field. The `init`(8) command verifies that `inittab` contains a line with the specified *id* field and that the *rstate* field includes the current run level. Then `init`(8) starts a login process on the specified terminal.

`initdefault` The `init`(8) command scans an entry with this action only when it is initially invoked; `init`(8) uses this entry, if it exists, to determine which run level to enter initially. It does this by taking the highest run level specified in the *rstate* field and using that as its initial state. If the *rstate* field is empty, the run level is interpreted as 0123456; `init`(8) enters run level 6. You can use the `initdefault` entry to specify that `init`(8) should start in the single-user state. If `init`(8) does not find an `initdefault` entry in `/etc/inittab`, it also requests an initial run level from the `/dev/syscon` terminal at reboot time.

`ldsynctm` Sets the `init ldsynctm` variable, which determines the system `ldsync` interval. The *process* field for this entry is specified in seconds. For details, see `ldsync`(8).

`off` When the process associated with this entry is currently running, `init`(8) sends the warning signal (`SIGTERM`) and waits 20 seconds before forcibly terminating the process by using the kill signal (`SIGKILL`). When the process is nonexistent, `init`(8) ignores the entry.

`once` On entering a run level that matches the *rstate* for the entry, `init`(8) starts the process and does not wait for its termination. When the process dies, `init`(8) does not restart it. If, on entering a new run level, the process is still running from a previous run-level change, the program will not be restarted.

`ondemand` — This instruction is really a synonym for the `respawn` action. It is functionally identical to `respawn`, but it is given a different keyword to divorce it from run levels. This is used only with the `a`, `b`, or `c` values discussed in the *rstate* field description.

`respawn` — If the process does not exist, `init`(8) will start the process; it will not wait for process termination (that is, it will continue to scan the `inittab` file). When the process dies, `init`(8) restarts it. If the process currently exists, `init`(8) will do nothing and will continue to scan the `inittab` file.

`sleeptime` — Sets the `init sleeptime` variable, which determines the system `sync` interval. The *process* field for this entry is specified in seconds. For details, see `sync`(1).

`sysinit` — The `init`(8) command executes entries of this type before trying to access the console. You should use this entry to initialize only the devices for which `init`(8) might ask for a run level; `init`(8) executes and waits for these entries before continuing.

`timezone` — Sets the systemwide local time zone. The contents of the *process* field are used to set the `TZ` environment variable. For a definition of the format for `TZ`, see `ctime`(3C). The `timezone` entry should follow the `initdefault` entry.

`wait` — On entering the run level that matches the *rstate* of the entry, `init`(8) starts the process and waits for its termination. While `init`(8) is in the same run level, all subsequent reads of the `inittab` file cause `init`(8) to ignore this entry.

process — An entry in this field is a `sh`(1) command to be executed. The `init`(8) command prefixes the entire *process* field with the `exec`(2) string and passes it to a forked `sh` process as `sh -c 'exec` *command*`'`. Therefore, any legal `sh` syntax can appear in the *process* field. You can insert comments with the #*comment* syntax.

EXAMPLES

The following is an example of an `inittab` entry for Minnesota in the central time zone in the U.S.A.:

```
tz::timezone:TZ=CST6CDT
```

The `init`(8) command passes this value to its children, they pass it to theirs, and so on, so that all processes interpret the time according to this `inittab` entry.

The `init sleeptime` and `ldsynctm` variables default to 30 seconds and 120 seconds, respectively. To reset these variables, create `inittab` entries such as the following:

```
st::sleeptime:60
lt::ldsynctm:180
```

The preceding `init`(8) commands reset the `sleeptime` and `ldsynctm` variables to 60 seconds and 180 seconds, respectively.

FILES

`/etc/initreq`	Pipe used when initiating a new login session
`/etc/inittab`	Script for `init`(8) process
`/usr/include/initreq.h`	Definition of request structure

SEE ALSO

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

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

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

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

General UNICOS System Administration, Cray Research publication SG–2301

NAME

inode – Inode format

SYNOPSIS

```
#include <sys/types.h>
#include <sys/param.h>
#include <sys/ino.h>
```

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

An inode for a regular file or a directory in a file system has a structure defined by the sys/ino.h include file.

The structure of an inode for NC1FS file systems is as follows:

```
struct cdinode {
        uint    cdi_rsrvd_1 : 8, /* Reserved for expansion of cdi_mode       */
                cdi_mode    :24, /* mode and type of file (4-bits still free)*/
                cdi_msref   : 1, /* Modification signature is referenced flag*/
                cdi_ms      :14, /* Modification signature                   */
                cdi_nlink   :17; /* #of links to file (can hold > 100,000)   */

        uint    cdi_rsrvd_2 : 8, /* Reserved for expansion of cdi_uid        */
                cdi_uid     :24, /* Owner's user-ID                          */
                cdi_rsrvd_3 : 8, /* Reserved for expansion of cdi_gid        */
                cdi_gid     :24; /* Owner's group-ID                         */

        uint    cdi_rsrvd_4 : 8, /* Reserved for expansion of cdi_acid       */
                cdi_acid    :24, /* Account-ID                               */
                cdi_gen     :32; /* Inode generation number                  */

        long    cdi_size;        /* Number of bytes in the file              */
        long    cdi_moffset;     /* Modification offset for current signature*/

        uint    cdi_blocks  :52, /* Quotas: #of blocks actually allocated    */
                cdi_extcomp : 1, /* Security: extended compartments flag     */
                cdi_secrsvd1:11; /* Security: reserved                       */
```

```
union {
        long smallcmps;  /* Compartments if [0..63]                   */
} cdi_compart;           /* Security: compartments info               */

uint    cdi_slevel  : 8, /* Security: security level                  */
        cdi_intcls  : 8, /* Security: integrity class (obsolete)      */
        cdi_secflg  :16, /* Security: flag settings                   */
        cdi_intcat  :32; /* Security: integrity category (obsolete)   */

long    cdi_permits;     /* Security: Permissions inherited at        */
                         /* execution time.                           */

union {
        daddr_t daddr;   /* Extent descriptor                         */
        dblk_t  dblk;    /* Block  descriptor                         */
} cdi_acl;               /* Security: ACL location                    */

uint    cdi_cpart   : 8, /* Next partition from cbits to use          */
        cdi_rsrvd_5 : 8, /* Reserved by the Kernel group.             */
        cdi_dmkey   :48; /* Data-Migration: key                       */

uint    cdi_allocf  : 4, /* Data-Block allocation flags               */
        cdi_alloc   : 4, /* Data-Block allocation technique           */
        cdi_cblks   :24, /* Number of blocks to allocate per part     */
        cdi_dmmid   :32; /* Data-Migration: machine-ID                */

uint    cdi_atmsec  :34, /* Access time (secs)                        */
        cdi_natmsec :30; /* Access time (nanosecs)                    */

uint    cdi_mtmsec  :34, /* Modification time (secs)                  */
        cdi_nmtmsec :30; /* Modification time (nanosecs)              */

uint    cdi_ctmsec  :34, /* Time of last inode modification (secs)     */
        cdi_nctmsec :30; /* Time of last inode modification (nanosecs)*/

long    cdi_cbits;       /* bit mask, file placement within cluster   */
```

```
        union {
                daddr_t daddr;   /* Extent descriptor                        */
                dblk_t  dblk;    /* Block  descriptor                        */
                long    whole;                 struct {
                       uint     one   :32,      /* half 1                     */
                                two   :32;      /* half 2                     */
                } half;                 struct {
                       uint     one   :16,      /* quarter 1                  */
                                two   :16,      /* quarter 2                  */
                                three :16,      /* quarter 3                  */
                                four  :16;      /* quarter 4                  */
                } quarter;                 struct {
                        uint    one   : 8,      /* eighth 1                   */
                                two   : 8,      /* eighth 2                   */
                                three : 8,      /* eighth 3                   */
                                four  : 8,      /* eighth 4                   */
                                five  : 8,      /* eighth 5                   */
                                six   : 8,      /* eighth 6                   */
                                seven : 8,      /* eighth 7                   */
                                eight : 8;      /* eighth 8                   */
                } eighth;
        } cdi_addr[8];           /* File allocation locators                 */
                                 /* The #define for NC1NADDR must not be > 8 */

        long    cdi_rsrvd[5];    /* Reserved by the Kernel group for use in  */
                                 /* future releases of UNICOS.               */
                                 /* No notification will be given when these */
                                 /* words will be employed by future versions*/
                                 /* of UNICOS.                               */

        long    cdi_slock[2];    /* Reserved for SFS lock structure          */

        long    cdi_sitebits;    /* Word reserved for site use.              */
};
```

FILES

/usr/include/sys/ino.h	Inode structure definition
/usr/include/sys/types.h	Data types definition file

SEE ALSO

fs(5), types(5)

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

NAME

ipc – Interprocess communication (IPC) access structure

SYNOPSIS

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

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4

DESCRIPTION

Three mechanisms use the sys/ipc.h include file for interprocess communication (IPC): messages, semaphores, and shared memory. All use a common structure type, ipc_perm, to pass information used in determining permission to perform an IPC operation.

The ipc_perm structure contains the following members:

uid_t	uid	Owner's user ID
gid_t	gid	Owner's group ID
uid_t	cuid	Creator's user ID
gid_t	cgid	Creator's group ID
mode_t	mode	Read/write permission

The uid_t, gid_t, mode_t, and key_t types are defined as described in sys/types.h.

Definitions are given for the following constants:

Mode bits:

IPC_CREAT	Creates entry if key does not exist.
IPC_EXCL	Fails if key exists.
IPC_NOWAIT	Returns an error if request must wait.

Keys:

IPC_PRIVATE	Specifies a private key.

Control commands:

IPC_GETACL	Gets access control list.
IPC_RMID	Removes identifier.
IPC_SET	Sets options.

`IPC_SETACL`	Sets access control list.
`IPC_SETLABEL`	Sets security label.
`IPC_STAT`	Gets options.

SEE ALSO

`msg`(5), `sem`(5), `shm`(5), `types`(5)

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

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

`ipc`(7) Online only

NAME

iptos – IP Type-of-Service database

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The /etc/iptos file contains the database of Type-of-Service (TOS) names used for the Internet Protocol (IP) TOS option.

Each entry must consist of a single line that contains the name of the TOS entry, the protocol name for which the entry is appropriate, the TOS value for the entry, and any aliases that exist for the entry. Items are separated by any number of blanks and/or tab characters. A # symbol indicates that the remaining portion of the line is a comment and is not interpreted by routines that search the file. Blank lines in the file are ignored.

TOS entry names may contain any printable character other than a blank, tab, new line, or comment (#).

A protocol name of * (one asterisk) indicates that the entry is valid for all protocols.

The TOS value for the entry must be a list of either symbolic names or numbers (octal, decimal, or hexadecimal) that correspond to TOS option bits or TOS precedence values, separated by **|** symbols. Recognized symbolic names for TOS option bits are as follows:

```
none                    0x00
delay                   0x10
throughput              0x08
reliability             0x04
reserved1               0x02
reserved2               0x01
```

Recognized symbolic names for TOS precedence values are as follows:

```
netcontrol              0xe0
internetcontrol 0xc0
crtic/ecp               0xa0
flashoverride   0x80
flash                   0x60
immediate               0x40
priority                0x20
routine         0x00
```

EXAMPLES

The following example shows typical entries in `/etc/iptos`:

```
#
# Format of this file:
# Application     Proto TOS-bits    aliases
#
# The Proto field may be "*" mean it doesn’t matter.
#
# For multiple values, use a "|", e.g, delay|throughput
#

delay            *      delay            lowdelay
reliability      *      reliability      highreliability
throughput       *      throughput       highthroughput

data             tcp    throughput       bulk-data batch rcp
data             udp    delay            bulk-data batch tftp

interactive      tcp    delay            rlogin telnet
interactive      udp    delay

bootp            *      none
domain   udp    delay           nameserver
domain   tcp    none            nameserver
egp      udp     none
ftp-control      tcp    delay
ftp-data         tcp    throughput
icmp-errors      icmp   none
icmp-queries icmp       none
igp              *      reliability      route router routed
nntp             *      none
smtp-cmd         tcp    delay
smtp-data        tcp    throughput
#smtp            tcp    none             # only if you can’t switch !!!
snmp             udp    reliability
tftp             udp    delay
```

FILES

`/etc/iptos` Contains names and TOS values

SEE ALSO

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

NAME

`issue` – Login message file

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `/etc/issue` file contains a message for interactive users that will be printed before the login prompt. The default login prompt is as follows:

```
login:
```

The `issue` file is an ASCII file that is read by `login`(1) and written to the terminal.

NOTES

Originally, `getty`(8) printed the message in `/etc/issue`; this occurred before `login`(1) executed during an interactive login. To facilitate network logins, this functionality has been duplicated in `login`(1).

FILES

`/etc/issue` Login message file

SEE ALSO

`inittab`(5)

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

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

NAME

`krb.conf` – Kerberos configuration file

SYNOPSIS

`/etc/krb.conf`

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `krb.conf` file contains configuration information that describes the Kerberos realm and the Kerberos key distribution center (KDC) servers for known realms. `krb.conf` contains the name of the local realm in the first line, followed by lines indicating realm and host entries. The first token is a realm name; the second is the host name of a host running a KDC for that realm. The words `admin server` following the host name indicate that the host also provides an administrative database server, as in the following example.

```
ATHENA.MIT.EDU
ATHENA.MIT.EDU kerberos-1.mit.edu admin server
ATHENA.MIT.EDU kerberos-2.mit.edu
LCS.MIT.EDU kerberos.lcs.mit.edu admin server
```

SEE ALSO

`krb.realms`(5)

`krb_get_krbhst`(3K), `krb_get_lrealm`(3K) in the *Kerberos User's Guide*, Cray Research publication SG–2409

NAME

`krb.realms` – Host to Kerberos realm translation file

SYNOPSIS

`/etc/krb.realms`

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `krb.realms` file provides a translation from a host name to the Kerberos realm name for the services provided by that host. For simple configurations in which only a single realm is being used, this file is not required.

Each line of the translation file is in one of the following forms:

host_name kerberos_realm

domain_name kerberos_realm

The *domain_name* field should be of the form `.XXX.YYY` (for example, `.LCS.MIT.EDU`).

If a host name exactly matches the *host_name* field in a line of the first form, the corresponding realm is the realm of the host. If a host name does not match any host name in the file, but its domain exactly matches the *domain_name* field in a line of the second form, the corresponding realm is the realm of the host.

If no translation entry applies, the realm of the host is considered to be the domain portion of the host name, converted to uppercase.

SEE ALSO

`krb_realmofhost`(3K) in the *Kerberos User's Guide*, Cray Research publication SG–2409

NAME

ldesc – Logical disk descriptor file

SYNOPSIS

```
#include  sys/ldesc.h
```

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

A logical disk descriptor file is used to combine one or more character or block special disk files to form one logical disk device. The ldesc structure in /usr/include/sys/ldesc.h, which defines the logical descriptor file, appears as follows:

```
struct  ldesc   {
        word    magic;
        word    nslices;                /* # of slices listed below */
        char    slice[64][48];          /* max 64  / logical device */
};

#define LDMAGIC         'LDMAGIC!'      /* magic word               */
```

The logical descriptor file can contain up to 64 absolute path names; each may consist of up to 48 characters. Each absolute path name is said to be "a member" or "a slice" of the logical disk device. The members are combined in a manner prescribed by the character or block special device logical device that references it.

To create a logical descriptor file, use the mknod(8) command, as follows:

mknod *name L member0* [*member1 member2* . . .]

FILES

/usr/include/sys/ldesc.h

SEE ALSO

dsk(4), ldd(4), pdd(4)

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

NAME

`license.dat` – License configuration file for FLEXlm licensed applications

SYNOPSIS

`/usr/local/flexlm/licenses/license.dat`

IMPLEMENTATION

All supported platforms

DESCRIPTION

The `license.dat` file contains the information that the flexible license manager (FLEXlm) network licensing package uses to determine the licenses that are available at a particular site. The `license.dat` file contains the list of server nodes, list of vendor daemons, and list of features enabled for the site. FLEXlm programs and routines find the license file by an algorithm described in the Finding the License File section of this man page.

The format of the license file is a server line (or lines), followed by one or more daemon lines, followed by one or more feature lines. The system administrator can change only the following four data items in the license file, allowing the administrator to configure the licensed software to fit into the environment:

- Node names on the server line(s)
- Port numbers on the server line(s)
- Path names on the daemon line(s)
- Options file path names on the daemon line(s)

The data in the license file is case-sensitive.

All other data in the license file is used to compute the encryption code, and you should enter it exactly as supplied by your software vendor.

Each line in the `license.dat` file starts with a keyword that identifies the information on that line. The keyword may be `SERVER`, `DAEMON`, or `FEATURE`. On unlimited node-locked features, such as UNICOS systems, server and daemon lines are not required.

Server Line

The server line specifies the node name and host ID of the license server and the port number of the license manager daemon (`lmgrd`). Usually, a license file has one server line; more than one server line indicates that you are using redundant servers.

The server line has the following form:

`SERVER` *nodename* *hostid* [*port-number*]

The server line accepts the following arguments:

nodename — Specifies the string returned by the UNICOS `hostname`(1) command. The system administrator can change the *nodename* field.

hostid — Specifies the string returned by the `lmhostid`(1) command. The host IDs from all of the server lines are encrypted into the feature lines; therefore, the system administrator cannot change `hostid`.

port-number — Specifies the TCP port number to use; if you omit this argument, the FLEXlm TCP service must be present in the network services database. The system administrator can change the optional *port-number* field at any time, which lets system administrators select a port number that does not conflict with the other services, software packages, or FLEXlm vendors on their system. The default port number for Cray Research licenses is 7169.

At sites that have multiple redundant servers, one of the servers is selected as the master node. If the order of the server lines is the same in the license files for all redundant servers, the first server in the list will be the master; otherwise, the server whose name is alphabetically first will be the master.

Daemon Line

The daemon line specifies the daemon name and path. The daemon line has the following form:

`DAEMON` *daemon-name pathname* [*options-file-pathname*]

The daemon line accepts the following arguments:

daemon-name — Specifies the name of the vendor daemon used to serve feature(s) in the file; the system administrator cannot change *daemon-name.*

pathname — Specifies the path name to the executable file for this daemon. System administrators can change the *pathname* field, which lets them place the vendor daemon in any convenient location.

options-file-pathname — Specifies the full path name of the end-user-specified options file for the daemon. FLEXlm does not require an options file. The system administrator can change the location of the options file, which describes various options that the system administrator can modify (see the `license.options`(5) man page).

Feature Line

The feature line describes the name of the feature to be licensed. A feature can be the name of a program, a program module, or option. Any amount of white space of any type (that is, tabs or spaces) can separate the components of a line. NOTE: The system administrator **cannot** change the information in the feature line.

The feature line has the following form:

`FEATURE` *name daemon version expdate nlic code* "*vendor_string*" [*hostid*]

The feature line accepts the following arguments:

name Specifies the name given to the feature by the vendor; the system administrator cannot change *name*.

daemon Specifies the name of the vendor daemon; also found in the daemon line. The specified daemon serves this feature. The system administrator cannot change *daemon*.

version Specifies the version of the feature this license supports; the system administrator cannot change *version*.

expdate Specifies the expiration date (for example, 7-may-1998). If the year is 0, the license never expires. The system administrator cannot change *expdate*.

nlic Specifies the number of concurrent licenses for the feature. If the number of users is set to 0, the licenses for the feature are uncounted and no `lmgrd` is required. The system administrator cannot change *nlic*.

code Specifies the encrypted password for the feature line. The start date is encoded into the code; thus, identical codes created with different start dates will be different. The system administrator cannot change *code*.

"*vendor_string*" Specifies the vendor-defined string, enclosed in double quotation marks. The string can contain any 64 characters, except a quotation mark (white space is ignored). The system administrator cannot change "*vendor_string*".

hostid Specifies the string returned by the `lmhostid`(1) command. *hostid* is used only if the feature will be bound to a particular host, whether or not its use is counted. Numeric *hostid*s are case-insensitive. The system administrator cannot change *hostid*.

Finding the License File

Most programs that read the `license.dat` file accept a command-line option (typically `-c`), which you can use to specify the location of the license file if it is not `/usr/local/flexlm/licenses/license.dat`. If you do not specify a command-line argument, the value of the `LM_LICENSE_FILE` environment variable will be used to find the license file. If you do not specify the option or the command-line argument, the default location, `/usr/local/flexlm/licenses/license.dat`, will be used.

You can use the `LM_LICENSE_FILE` environment variable to specify as many different license files as needed. To do this, you should set the environment variable to one string that contains all of the license file paths separated by colons. The following is an example, using the `csh` shell:

```
setenv LM_LICENSE_FILE /usr/local/foo.dat:/u2/flexlm/bar.dat:/u12/lic.dat
```

EXAMPLES

An example of a `license.dat` file follows; it illustrates the license file for one vendor that has two features and a set of three server nodes, any two of which must be running for the system to function:

```
SERVER pat   3e9 7169
SERVER lee   1fb 7169
SERVER terry 2a3 7169
DAEMON craylmd /etc/craylm
FEATURE great_program craylmd 1.000 01-jan-1995 10 1EF890030EABF324 ""
FEATURE greater_program craylmd 1.000 01-jan-1995 10 0784561FE98BA073 ""
```

An example of a `license.dat` file for unlimited node-locked features follows:

```
FEATURE nqs_nl none 1.000 1-jul-95 0 AWQHG947YUN548E390DF "" 3e9
FEATURE nqs_fl none 1.000 1-jul-95 0 AW3HG930YUN5EOE3W7PX "" 3e9
FEATURE nqx none 1.000 1-jul-95 0 PJM693SE27XGF860GV09 "" 3e9
FEATURE onc none 1.000 1-jul-95 0 2QDTY572T09KM114BL90 "" 3e9
FEATURE dfs_c none 1.000 1-jul-95 0 RD2CP3IOGON8206JU73A "" 3e9
FEATURE dfs_s none 1.000 1-jul-95 0 SJS1046LAP0213KOMXX5 "" 3e9
FEATURE sfs none 1.000 1-jul-95 0 YAPA02947NUKE330TQ21 "" 3e9
FEATURE tsr none 1.000 1-jul-95 0 29BHS90EOJ83XZ4UP07W "" 3e9
FEATURE HEXAR none 1.000 1-jul-95 0 70BCT04MV3DE2NJU00L3 "" 3e9
```

FILES

`/usr/local/flexlm/licenses/license.dat`
Default location of license configuration file for FLEXlm licensed applications

SEE ALSO

`lmgrd`(1) for information about starting up FLEXlm license daemons

`license.options`(5) for information about the system administrator options file for FLEXlm licensed applications in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

NAME

`license.options` – System administrator options file for FLEXlm licensed applications

SYNOPSIS

`/usr/local/flexlm/options/license.opt`

IMPLEMENTATION

All supported platforms

DESCRIPTION

The `license.opt` file contains optional flexible license manager (FLEXlm) information supplied by the system administrator at the end-user site. You can use this information to tailor the behavior of the license daemons. The options file can contain the following information:

- Reserved license information
- Log file control options
- License time-out control
- License access control

Lines that begin with a # are ignored, and you can use them as comments.

No default location or name for the options file exists; it is active only if it has been specified in the `license.dat` file as the fourth argument on the daemon line. If multiple daemon lines are in the `license.dat` file, multiple options files can exist, one for each daemon line.

Each line in the options file controls one option; each line starts with a keyword (`EXCLUDE`, `EXCLUDEALL`, `GROUP`, `INCLUDE`, `INCLUDEALL`, `NOLOG`, `RESERVE`, or `TIMEOUT`) that identifies the information on that line. Not all of the lines in an options file refer to a feature; therefore, to use the nolog line, the system administrator must set up separate options files.

Reserve Line

The reserve line reserves licenses for a user; it has the following form:

`RESERVE` *numlic featurename type reservename*

The reserve line accepts the following arguments:

numlic Specifies the number of licenses to reserve.

featurename Specifies the feature to reserve.

type Specifies the type of user for which to reserve licenses; *type* may be `GROUP`, `USER`, `HOST`, or `DISPLAY`.

reservename Specifies the name of the user or group for which to reserve licenses.

Any licenses reserved for a use are dedicated to that user; even when that user is not actively using the license, it will be unavailable to other users.

Nolog Line

The nolog line turns off logging of specific events from the `lmgrd`(1) command. Specifying a nolog line reduces the amount of output to the log file, which can be useful in those cases in which the log file grows too quickly. The nolog line has the following form:

`NOLOG` *what*

The nolog line accepts the following argument:

what Specifies what to turn off; *what* may be `IN` (checkins), `OUT` (checkouts), `DENIED` (denied requests), or `QUEUED` (queued requests).

Group Line

The group line defines collections of users, which you can then use in reserve, include, or exclude lines. The group line has the following form:

`GROUP` *groupname usernamelist*

The group line accepts the following arguments:

groupname Specifies the name of the group being defined.

usernamelist Specifies the list of user names in that group.

In the FLEXlm v3.0 release, multiple group lines adds all of the users specified into the group; before the FLEXlm v3.0 release, daemons do not allow multiple group lines to concatenate.

Include and Exclude Lines

The include and exclude lines specify a user, host, display, group of users, or Internet addresses in the list of users who are allowed (on include line) or not allowed (on exclude line) to use the feature. Specifying an include line has the effect of excluding everyone else from that feature; thus, only those users specified in the include line for a specified feature can use that feature. Any user specified in the exclude line for a specified feature cannot use that feature.

The include and exclude lines have the following form:

`[INCLUDE | EXCLUDE]` *feature type name*

The include and exclude lines accept the following arguments:

feature Specifies the name of the feature being affected.

type Specifies the type to be included or excluded; *type* may be `USER`, `HOST`, `DISPLAY`, `GROUP`, or `INTERNET`.

name Specifies the name of the user or group to include or exclude.

Includeall and Excludeall Lines

The includeall and excludeall lines specify which users, hosts, displays, groups, or Internet addresses can use all features that this daemon supports. Specifying an includeall line has the effect of excluding everyone else from all features; thus, only those users specified in the includeall line can use the daemon's features. Any user specified in the excludeall line cannot use any of the features that this daemon supports.

The includeall and excludeall lines have the following form:

`[INCLUDEALL | EXCLUDEALL]` *type name*

The includeall and excludeall lines accept the following arguments:

type Specifies the type to be included or excluded; *type* may be `USER`, `HOST`, `DISPLAY`, `GROUP`, or `INTERNET`.

name Specifies the name of the user or group to include or exclude.

The Internet address is specified in the standard IP address notation, and parts of the address can be wildcarded with a * symbol. An example is as follows:

```
192.9.200.1
192.9.200.*
```

For example, the following line would allow only users from the 192.9.200 network to use the features of this daemon; any users from machines on another network would not have access to these features:

```
INCLUDEALL INTERNET 192.9.200.*
```

The includeall and excludeall lines and the `INTERNET` type are available only in the FLEXlm v2.4 release or later.

Timeout Line

The timeout line sets up a minimum idle time after which a user will lose the license if it is not in use. Using this line allows the system administrator to prevent users from wasting a license (by keeping it checked out when users are not using it) when someone else wants a license. The timeout line has the following form:

`TIMEOUT` *feature idletime*

The timeout line accepts the following arguments:

feature Specifies the name of the feature.

idletime Specifies the number of seconds after which an inactive license is reclaimed. If you do not specify a time-out value (*idletime*) in your options file, no time-out exists for that feature.

EXAMPLES

An example of an options file follows:

```
RESERVE 1 f1 USER pat
RESERVE 1 f1 USER less
RESERVE 1 f1 HOST terry
NOLOG QUEUED
INCLUDE f1 USER bob
EXCLUDE f1 USER hank
INCLUDEALL USER sallie
EXCLUDEALL HOST chaos
GROUP Hackers bob howard james
TIMEOUT f1 3600
```

FILES

`/usr/local/flexlm/options/license.opt`
Default location of system administrator options file for FLEXlm licensed applications

SEE ALSO

`lmgrd`(1) for information about starting up FLEXlm license daemons

`license.dat`(5) for information about the license configuration file for FLEXlm licensed applications in the *UNICOS File Formats and Special Files Reference Manual*, Cray Research publication SR–2014

NAME

`lnode` – Kernel user limits structure for fair-share scheduler

SYNOPSIS

`#include <sys/lnode.h>`

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

The fair-share scheduler uses the kernel `lnode` structure to maintain per-user resource limits while a user has processes running. The `login`(1) command establishes the lnodes by using the `limits`(2) system call when a new user logs in to the system. An lnode is *dead* when the last process attached to that lnode exits. `shrdaemon`(8) removes dead lnodes.

The kernel maintains a table of entries that contains per-user resource limits information. The `kern_lnode` structure defines each entry in the kernel's table. The fair-share scheduler uses this information to calculate and check limits for processes run by active users.

Within each `kern_lnode` structure entry is a subentry defined by the `lnode` structure. The subentry contains information that the kernel maintains and stores for all users (active and inactive). Therefore, the structure of a user's lnode depends on whether that user is active. An active user's `lnode` structure is defined in the `kern_lnode` structure and is located in the kernel `lnode` table. An inactive user's `lnode` structure is defined in the `lnode` structure and is stored in the file system.

An lnode is defined in the `sys/lnode.h` include file; the structure `lnode` includes the following elements:

```
struct lnode {
       char     l_name[16];     /* system-wide unique user name          */
       int      l_uid;          /* real uid for owner of this node       */
       int      l_group;        /* uid for this node's scheduling group  */
       long     l_flags;        /* flags                                 */
       short    l_shares;       /* allocated shares                      */
       short    l_plimit;       /* max # of processes allowed            */
       mlimit_t l_mlimit;       /* max clicks usable by all procs        */
       time_t   l_cpu_used;     /* used cpu budget in hertz              */
       time_t   l_cpulimit[L_NLIMTYPES];
                                /* total cpu budget in hertz for abs,    */
                                /* hard and soft                         */
       int      l_hcpuaction;   /* hard cpu action: terminate, chkpnt    */
       time_t   l_lowcpulval;   /* lowest cpu limit value                */
       int      l_lowcputype;   /* lowest cpu limit type: abs, hard, soft */
       long     l_reserved[2];  /* Reserved                              */
       float    l_usage;        /* decaying accumulated costs            */
       float    l_charge;       /* long term accumulated costs           */
};
```

The following flags are defined in the `l_flags` field. Knowledge of these flags can be useful when examining output from the `crash`(8) and `shrtree`(8) commands.

```
#define LASTREF           020  /* set for L_DEADLIM if last reference to     */
                               /*  this lnode                                */
#define ACTIVELNODE    010000  /* this lnode is on active list               */
#define CHNGDLIMITS    020000  /* this lnode's limits have changed           */
#define NOTSHARED      040000  /* this lnode does not get a share of the m/c */
#define DEFERTORESGRP 0100000  /* use l_group for this user's lnode          */
#define SHAREHOLDER  01000000  /* Defines UDB entry for nesting share levels */
```

The `l_charge` field comes from the `shcharge` field in the UDB; it is the long-term accumulated charge for consumption of resources. For group leaders, it represents the charge for the whole group.

The `l_usage` field comes from the `shusage` field in the UDB; it represents recent usage of resources. The scheduler uses this field to determine whether processes that the lnode owns are entitled to CPU resources.

An lnode is part of the kernel lnode (`kern_lnode`) structure. The kernel lnode structure holds temporary values that the scheduler uses, as well as static values associated with the user. The `kern_lnode` structure contains the following fields:

```
typedef struct kern_lnode *      KL_p;

struct kern_lnode {
       KL_p         kl_next;     /* next in active list             */
       KL_p         kl_prev;     /* prev in active list             */
       KL_p         kl_parent;   /* group parent                    */
       KL_p         kl_gnext;    /* next in parent's group          */
       KL_p         kl_ghead;    /* start of this group             */
       struct lnode kl;          /* the limits (as above)           */
       float        kl_gshares;  /* total shares for this group     */
       float        kl_eshare;   /* effective share for this group  */
       float        kl_norms;    /* normalised shares for lnode     */
       float        kl_usage;    /* kl.l_usage / kl_norms           */
       float        kl_totuse;   /* sum of 1/usage                  */
       float        kl_rate;     /* active process rate for lnode   */
       float        kl_temp;     /* temporary for scheduler         */
       int          kl_cost;     /* cost accumulating in            */
                                 /* current period                  */
       float        kl_rshare;   /* current dynamic machine share   */
       int          kl_cpu;      /* unweighted CPU clicks (OS_HZ)   */
       int          kl_muse;     /* actual number of pages used     */
       int          kl_refcount; /* processes attached to this lnode*/
       int          kl_children; /* lnodes attached to this lnode   */
       float        kl_nrun;     /* runnable proc's on this lnode   */
       float        kl_adj;      /* adjustment factor (adjgroups)   */
};
```

The `kern_lnode` structures in the kernel table are grouped together in a tree. At any level in the tree, the share of resources allocated to an individual lnode is that proportion of the group's resources represented by the ratio of the lnode's shares to the total shares of all lnodes in the group. The `l_group` field represents the user ID of the parent lnode for an individual lnode. (All lnodes in a group have the same parent lnode). The root's lnode, which is initialized at system boot time, represents the top of the tree. An lnode used by all idle processes also is started at boot time with a 0% share of the machine.

When the last process referencing the lnode has exited, the LASTREF bit in `l_flags` is set for use by the `limits`(2) system call. If this lnode was the last one referencing its group, the DEADGROUP bit is set. The `limits`(2) system call collects dead groups.

The scheduling priority of each running process is recalculated each minor clock tick by the following method. The recent usage (kl_usage) for the process is multiplied by the user's active process rate (kl_rate) and the result is added to the scheduling priority for the process in the structure of the process. This scheduling priority value decays by an amount that depends on the nice value for the process (the lower the priority of the process, the slower the decay). The scheduling priority is copied to the structure of the process for use by the UNICOS low-level scheduler. (The low-level scheduler recalculates the priority of each nonrunning process by using this value.)

FILES

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

SEE ALSO

share(5)

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

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

UNICOS Resource Administration, Cray Research publication SG–2302

NAME

mailrc, mailx.rc – Start-up files for mailx(1)

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The /usr/lib/mailx/mailx.rc and .mailrc files are start-up files for the UNICOS mail program mailx(1). When mailx(1) is invoked, it reads commands from a system file, /usr/lib/mailx/mailx.rc, to initialize certain parameters and variables on a systemwide basis. The mailx(1) program then looks in your home directory for a file called .mailrc; if .mailrc exists, mailx(1) reads in the commands in that file.

Most mailx(1) commands are legal inside a start-up file. The most useful and appropriate commands for inclusion in the .mailrc file modify the display, disposition, and sending of messages. A list of such commands follows (for a complete list and description of all valid commands, see mailx(1)):

#	Comment line; mailx(1) ignores the rest of the line.
alias	Declares an alias.
alternates	Lists alternative account names that can access your mail.
discard	Suppresses printing of specified header fields in messages.
group	Declares a group.
if	Allows conditional processing (with else and endif).
ignore	Suppresses printing of specified header fields in messages.
mbox	Specifies a file for storage of read messages.
set	Sets mailx(1) environment variables.
unset	Clears mailx(1) environment variables.
version	Prints the version of the mailx program.

The following mailx(1) commands are not legal in a start-up file: !, Copy, edit, followup, Followup, hold, mail, preserve, reply, Reply, shell, and visual. If any of these commands occurs in a start-up file, the remaining lines in the file are ignored. For a description of these commands, see mailx(1).

NOTES

Any errors in a start-up file cause the remaining lines in the file to be ignored.

EXAMPLES

Example 1: The following is an example of a typical `/usr/lib/mailx/mailx.rc` file:

```
# Use sendmail to deliver mail
set sendmail=/usr/lib/sendmail
```

Example 2: The following is an example of a typical `.mailrc` file:

```
# Append messages to the end of $HOME/mbox
set append
# Ask for a subject line when sending mail
set asksub header
# Enable printing of header information when reading mail
set header
# Set screen size to 20 lines
set crt=20
# Use more to paginate long messages
set PAGER=more
# Store a copy of mail I send in outgoing.mail
set record=$HOME/outgoing.mail
# Don't display certain fields in messages
ignore Received Date Message-Id In-Reply-To Status
```

FILES

`$HOME/.mailrc`	Personal start-up file
`$HOME/mbox`	Secondary storage file
`/tmp/R[emqsx]*`	Temporary files for `mailx`
`/usr/lib/mailx/mailx.help*`	`mailx`(1) help message files
`/usr/lib/mailx/mailx.rc`	Systemwide start-up file
`/usr/mail/*`	Primary storage files

SEE ALSO

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

NAME

MAKEFILE – File containing site–specific make(1) rules

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

The /etc/MAKEFILE file can be created by a site's system administrator and used to define make(1) rules for an environment specific to the site. This file is controlled by the site system administrator, and its content is unrestricted: anything allowed for a user makefile can be in the /etc/MAKEFILE file. /etc/MAKEFILE must have world read permission.

Typical uses for /etc/MAKEFILE would be to add new suffix rules, to modify built–in make(1) default rules, or to specify special targets. For example, if /etc/MAKEFILE contains the special target .POSIX, the make(1) built–in default POSIX rules are used. If /etc/MAKEFILE contains the special target .SUFFIXES (without parameters), all make(1) default suffix rules are ignored, and /etc/MAKEFILE can define a new set of suffix rules.

The /etc/MAKEFILE file functions much like the make(1) include file. The /etc/MAKEFILE file is read as the first file when make(1) is invoked, before any of the user makefiles are processed.

Users may need to ignore /etc/MAKEFILE if their preexisting makefiles do not work with the rules defined in /etc/MAKEFILE. You can ignore /etc/MAKEFILE by using the -l option of make(1).

EXAMPLES

The following is an example /etc/MAKEFILE that specifies the following site-specific rules:

- Require POSIX rules (.POSIX:)
- Execute commands but do not echo them (.SILENT:)
- Add new suffixes (.u.v)
- Change existing suffixes (.c.o)

```
        .POSIX:
        .SILENT:
        .SUFFIXES: .u .v
.u.v:
        echo target $* from $<
.c.o:
        $(CC) -c $< -o $@
```

FILES

`/etc/MAKEFILE` Contains site–specific default `make` rules

SEE ALSO

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

NAME

masterfile – Internet domain name server master data file

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

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

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

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

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

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

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

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

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

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

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

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

EXAMPLES

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

```
$ORIGIN             ourdomain.com

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

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

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

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

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

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

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

FILES

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

SEE ALSO

`named.boot`(5)

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

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

NAME

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

SYNOPSIS

`/etc/mib.txt`

`/etc/snmpd.defs`

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

FILES

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

SEE ALSO

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

RFCs 1155, 1212, 1213

NAME

mnttab – Mounted file system table format

SYNOPSIS

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

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

The fields in the mnttab structure have the following meanings:

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

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

mt_ro_flg — Mounted device's read and write permissions.

mt_time — Date the device was mounted.

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

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

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

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

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

FILES

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

SEE ALSO

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

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

General UNICOS System Administration, Cray Research publication SG–2301

NAME

`motd` – File that contains message of the day

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

FILES

`/etc/motd` Message of the day file

SEE ALSO

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

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

NAME

msg – Message queue structures

SYNOPSIS

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

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

The following data types are defined through typedef:

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

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

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

The following message operation flag can be specified:

MSG_NOERROR

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

The msqid_ds structure contains the following members:

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

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

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

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

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

SEE ALSO

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

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

`ipc`(7) Online only

NAME

`named.boot` – Domain name server configuration file

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

NOTES

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

EXAMPLES

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

```
directory  /usr/domains

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

cache       .                            cache

primary    0.0.127.IN-ADDR.ARPA          local

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

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

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

FILES

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

SEE ALSO

`masterfile`(5)

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

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

NAME

`netgroup` – List of network groups

SYNOPSIS

`/etc/netgroup`

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

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

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

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

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

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

NOTES

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

WARNINGS

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

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

SEE ALSO

`exports`(5)

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

NAME

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

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

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

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

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

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

CAUTIONS

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

EXAMPLES

An example of a `.netrc` file follows:

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

        macdef pwdlsf
        pwd
        ls -CF

machine blackhole login anonymous password bonnie
```

FILES

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

SEE ALSO

`hosts`(5)

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

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

NAME

networks – Network name database

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

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

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

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

EXAMPLES

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

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

FILES

`/etc/networks` Contains network information

SEE ALSO

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

NAME

nl_types – Defines message system variables

SYNOPSIS

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

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

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

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

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

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

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

SEE ALSO

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

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

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

NAME

passwd – Format of the password file

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

Each entry contains the following fields:

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

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

FILES

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

SEE ALSO

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

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

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

NAME

printcap – Printer capability database

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

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

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

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

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

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

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

The following table shows printer capabilities:

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

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

NOTES

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

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

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

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

The same process applies to the xc and xs fields.

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

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

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

EXAMPLES

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

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

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

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

SEE ALSO

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

NAME

profile – Format of shell start-up file

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

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

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

EXAMPLES

An example of a typical .profile file follows:

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

FILES

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

SEE ALSO

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

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

NAME

proto – Prototype job file for at(1)

SYNOPSIS

/usr/lib/cron/.proto

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

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

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

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

EXAMPLES

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

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

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

FILES

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

SEE ALSO

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

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

`environ`(7) (available only online)

NAME

protocols – Protocol name database

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

EXAMPLES

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

```
# Internet (IP) protocols

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

FILES

/etc/protocols	Contains information about known protocols

SEE ALSO

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

NAME

publickey – Public key database

SYNOPSIS

/etc/publickey

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

SEE ALSO

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

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

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

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

NAME

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

SYNOPSIS

/usr/lib/cron/queuedefs

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

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

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

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

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

Empty fields are initialized to the default values.

Lines that begin with # are comments and are ignored.

EXAMPLES

The following is an example of a queuedefs file:

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

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

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

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

FILES

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

SEE ALSO

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

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

NAME

quota – Quota control file format

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

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

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

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

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

Quota File Header

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

The qf_header structure is defined as follows:

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

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

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

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

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

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

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

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

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

qf_qfnamesize
: Size (in bytes) of qf_name.

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

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

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

The `q_header` structure is defined as follows:

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

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

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

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

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

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

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

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

Quota Control Structures

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

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

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

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

```
(ID % 2039)
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

FILES

`sys/quota.h` Quota control definition file

SEE ALSO

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

UNICOS Resource Administration, Cray Research publication SG–2302

NAME

relo – Relocatable object table format under UNICOS

SYNOPSIS

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

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

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

The relocatable binary tables are presented as C language code.

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

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

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

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

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

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

Schematic Representations

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

Table header word

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

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

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

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

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

Program Descriptor Table (PDT)

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

PDT header

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

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

PDT block name

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

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

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

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

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

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

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

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

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

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

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

PDT Entry Point

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

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

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

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

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

PDT External

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

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

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

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

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

Text Table (TXT)

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

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

The text words immediately follow the item header.

Relocation Table (REL)

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

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

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

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

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

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

Extended Relocation Table (XRL)

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

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

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

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

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

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

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

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

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

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

Module Termination Table (MTT)

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

The MTT is defined by the following structure.

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

FILES

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

SEE ALSO

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

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

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

NAME

`resolv.conf` – Domain name resolver configuration file

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

The configuration options are as follows:

`domain local.domain`

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

`nameserver address`

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

`options option`

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

`debug` Sets `RES_DEBUG` in `res.options`.

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

`search search.list`

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

`sortlist address.list`

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

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

```
sortlist 130.155.160.0/255.255.240.0 130.155.0.0
```

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

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

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

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

EXAMPLES

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

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

FILES

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

SEE ALSO

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

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

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

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

NAME

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

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

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

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

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

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

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

remote_host
or
remote_host *remote_account_name*
or
*

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

NOTES

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

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

MESSAGES

The following error message may occur:

`permission denied`

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

SEE ALSO

`hosts.equiv`(5)

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

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

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

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

NAME

`rmtab` – List of remotely mounted file systems

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

hostname`:`*directory*

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

BUGS

The `rmtab` table is not always completely accurate.

FILES

`/etc/rmtab`	List of remotely mounted file systems

SEE ALSO

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

NAME

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

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

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

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

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

  `@h`*DDDDD*

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

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

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

Each of these entries is described as follows:

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

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

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

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

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

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

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

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

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

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

`@f` *flag* *optional text*

The flags are defined as follows:

`@ft` *type of program*
: Defines the replacement for the `%Y%` identification keyword.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

`@I` *DDDDD*
Insert

`@D` *DDDDD*
Delete

`@E` *DDDDD*
End

SEE ALSO

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

NAME

sectab – Format for table of defined security names and values

SYNOPSIS

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

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

```
#define MAXNAMES 64
#define MAXNAMELEN 256

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

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

FILES

/usr/include/sys/sectab.h

SEE ALSO

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

NAME

sem – Semaphore facility

SYNOPSIS

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

IMPLEMENTATION

All Cray Research systems

STANDARDS

POSIX, XPG4

DESCRIPTION

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

The following semaphore operation flag can be specified:

SEM_UNDO Sets up adjust on exit entry.

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

GETNCNT Gets semncnt.

GETPID Gets sempid.

GETVAL Gets semval.

GETALL Gets all cases of semval.

GETZCNT Gets semzcnt.

SETVAL Sets semval.

SETALL Sets all cases of semval.

The semid_ds structure contains the following members:

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

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

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

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

The structure `sembuf` contains the following members:

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

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

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

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

SEE ALSO

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

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

`ipc`(7) Online only

NAME

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

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

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

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

FILES

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

SEE ALSO

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

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

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

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

NAME

services – Network service name database

SYNOPSIS

/etc/services

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

TCP/IP entries:

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

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

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

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

EXAMPLES

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

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

FILES

`/etc/services` Network database file

SEE ALSO

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

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

NAME

share – Fair-share scheduler parameter table

SYNOPSIS

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

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

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

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

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

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

struct sh_consts
{
     /** Parameters **/

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

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

          /** Feedback **/

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

      #ifdef     KERNEL
      extern struct sh_consts     shconsts;
      #endif

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

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

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

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

      extern float          NiceDecays[];

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

      extern float          NiceRates[];

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

      extern int          NiceTicks[];
      #endif
```

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

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

FILES

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

SEE ALSO

`lnode`(5)

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

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

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

UNICOS Resource Administration, Cray Research publication SG–2302

NAME

shells – List of available user shells

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

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

EXAMPLES

An example of a /etc/shells file follows:

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

FILES

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

SEE ALSO

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

NAME

shm – Shared memory facility

SYNOPSIS

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

IMPLEMENTATION

CRAY T90 series

STANDARDS

POSIX, XPG4

DESCRIPTION

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

`SHM_RDONLY`	Attaches read-only (defaut is read-write).
`SHM_RND`	Rounds attach address to `SHMLBA`.
`SHMLBA`	Specifies segments low boundary address multiple.

The following data types are defined through `typedef`:

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

The `shmid_ds` structure contains the following members:

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

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

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

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

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

SEE ALSO

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

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

`ipc`(7) Online only

NAME

slrec – Security log record format

SYNOPSIS

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

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

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

The expanded header is defined as follows:

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

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

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

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

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

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

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

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

};
```

FILES

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

SEE ALSO

slog(4)

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

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

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

General UNICOS System Administration, Cray Research publication SG–2301

NAME

symbol – UNICOS symbol table entry format

SYNOPSIS

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

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

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

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

FILES

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

SEE ALSO

a.out(5), relo(5)

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

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

NAME

tapereq – Tape daemon interface definition file

SYNOPSIS

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

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

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

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

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

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

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

FILES

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

SEE ALSO

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

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

Tape Subsystem Administration, Cray Research publication SG–2307

NAME

`tapetrace` – Tape daemon trace file format

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

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

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

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

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

FILES

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

SEE ALSO

`tpdaemon`(8)

Tape Subsystem Administration, Cray Research publication SG–2307

NAME

`tar` – Tape archive file format

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

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

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

The default header block is as follows:

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

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

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

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

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

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

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

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

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

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

BUGS

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

SEE ALSO

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

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

NAME

`taskcom` – Task common table format

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

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

```
 -------------------------------------------------------------
| version | unused | nblks  |            tlen                 |
|   (7)   |  (9)   |  (16)  |            (32)                 |
 -------------------------------------------------------------
```

`version` `$TASKCOM` version ID (the value is 1)

`nblks` Number of task common blocks

`tlen` Total length of all task common blocks

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

```
 -------------------------------------------------
|         blen          |         offset          |
|         (32)          |          (32)           |
 -------------------------------------------------
| ival | unused | nlen |          nameptr         |
|  (1) |  (23)  |  (8) |            (32)          |
 -------------------------------------------------
|                    preset                       |
|                     (64)                        |
 -------------------------------------------------
```

`blen` Number of words in this block.

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

`ival` Block initialization flag:

0 No initialization

1 Initialization

This flag is currently unused.

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

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

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

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

```
 -----------------------
|         name          |
 -----------------------
|         name          |
 -----------------------
|          .            |
|          .            |
|          .            |
 -----------------------
```

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

SEE ALSO

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

NAME

`term` – Format of compiled `term` file

SYNOPSIS

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

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

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

The –2 value is represented by the following:

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

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

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

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

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

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

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

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

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

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

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

NOTES

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

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

EXAMPLES

The following are examples of compiled `term` files:

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

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

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

FILES

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

SEE ALSO

`terminfo`(5)

`curses`(3) (available only online)

`term`(7) (available only online)

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

NAME

terminfo – Terminal capability database

SYNOPSIS

/usr/lib/terminfo/?/*

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

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

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

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

Capabilities

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

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

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

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

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

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

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

Variable	Name	Code	Description
Booleans:			
`auto_left_margin`	`bw`	`bw`	`cub`1 wraps from column 0 to last column.
`auto_right_margin`	`am`	`am`	Terminal has automatic margin.
`no_esc_ctlc`	`xsb`	`xb`	Beehive (f1=< ESCAPE/ >, f2=< CONTROL-C/ >).
`ceol_standout_glitch`	`xhp`	`xs`	Standout not erased by overwriting (hp).
`eat_newline_glitch`	`xenl`	`xn`	New line ignored after 80 columns (Concept).
`erase_overstrike`	`eo`	`eo`	Can erase overstrikes with a blank.
`generic_type`	`gn`	`gn`	Generic line type (for example, dialup, switch).
`hard_copy`	`hc`	`hc`	Hard-copy terminal.
`hard_cursor`	`chts`	`HC`	Cursor is hard to see.
`has_meta_key`	`km`	`km`	Has a meta key (shift, sets parity bit).
`has_status_line`	`hs`	`hs`	Has extra "status line."
`insert_null_glitch`	`in`	`in`	Insert mode distinguishes nulls.
`memory_above`	`da`	`da`	Display may be retained above the screen.
`memory_below`	`db`	`db`	Display may be retained below the screen.
`move_insert_mode`	`mir`	`mi`	Safe to move while in insert mode.
`move_standout_mode`	`msgr`	`ms`	Safe to move in standout modes.
`needs_xon_xoff`	`nxon`	`nx`	Padding will not work, `xon`/`xoff` required.
`non_rev_rmcup`	`nrrmc`	`NR`	`smcup` does not reverse `rmcup`.
`no_pad_char`	`npc`	`NP`	Pad character does not exist.
`over_strike`	`os`	`os`	Terminal overstrikes on hard-copy terminal.
`prtr_silent`	`mc5i`	`5i`	Printer does not echo on screen.
`status_line_esc_ok`	`eslok`	`es`	Escape can be used on the status line.
`dest_tabs_magic_smso`	`xt`	`xt`	Destructive tabs, magic `smso` character (t1061).
`tilde_glitch`	`hz`	`hz`	Hazeltine; cannot print tildes(˜).
`transparent_underline`	`ul`	`ul`	Underline character overstrikes.
`xon_xoff`	`xon`	`xo`	Terminal uses `xon`/`xoff` handshaking.
Numbers:			
`columns`	`cols`	`co`	Number of columns in a line.

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

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

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

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

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

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

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

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

Sample Entry

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

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

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

Types of Capabilities

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

Finally, string-valued capabilities, such as `el` (clear to end of line sequence) are given by the two- to five-character capname, an =, and then a string ending at the next following comma. A delay in milliseconds may appear anywhere in such a capability, enclosed in `$<..>` angle brackets, as in `el=\EK$<3>`, and padding characters are supplied by `tputs`() [see `curses`(3)] to provide this delay. The delay can be either a number, for example, 20, or a number followed by an * (that is, `3*`), a `/` (that is, `5/`), or both (that is, `10*/`). An * symbol indicates that the padding required is proportional to the number of lines affected by the operation, and the amount given is the per-affected-unit padding required. (In the case of insert character, the factor is still the number of lines affected. This is always one unless the terminal has `in` and the software uses it.) When you specify a *, it is sometimes useful to give a delay of the form `3.5` to specify a delay per unit to tenths of milliseconds. (Only one decimal place is allowed.) An / symbol indicates that the padding is mandatory. Otherwise, if the terminal has `xon` defined, the padding information is advisory and will be used only for cost estimates or when the terminal is in raw mode. Mandatory padding will be transmitted regardless of the setting of `xon`.

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

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

Preparing Descriptions

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

Basic Capabilities

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

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

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

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

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

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

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

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

The Lear Siegler ADM-3 is described as follows:

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

Parameterized Strings

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

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

The `%` encodings have the following meanings:

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

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

If-then-else, %e elsepart is optional; else-if's are possible as in the Algol 68 language:

%? c_1 %t b_1 %e c_2 %t b_2 %e c_3 %t b_3 %e c_4 %t b_4 %e b_5%;

c_i are conditions, b_i are bodies.

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

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

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

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

Cursor Motions

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

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

Area Clears

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

Insert/Delete Line

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

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

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

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

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

Insert/Delete Character

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

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

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

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

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

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

Highlighting, Underlining, and Visible Bells

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Keypad

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

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

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

Tabs and Initialization

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

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

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

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

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

Delays

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

Status Lines

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

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

Line Graphics

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

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

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

Example:

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

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

Miscellaneous

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

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

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

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

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

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

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

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

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

Special Cases

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

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

Terminals that ignore a line feed immediately after an `am` wrap, such as the Concept 100, should indicate `xenl`. Those terminals whose cursor remains on the rightmost column until another character has been received, rather than wrapping immediately on receiving the rightmost character, such as the VT100, also should indicate `xenl`.

If `el` is required to remove standout (instead of writing normal text on top of it), you should specify `xhp`.

Those Teleray terminals whose tabs turn all characters moved over to blanks, should indicate `xt` (destructive tabs). This capability also is taken to mean that it is not possible to position the cursor on top of a ''magic cookie''; therefore, to erase standout mode, use delete and insert line.

Those Beehive Superbee terminals that do not transmit the escape or `<CONTROL-C>` characters, should specify `xsb`, indicating that the <`f1`> key will be used for escape and the <`f2`> key for `<CONTROL-C>`.

Similar Terminals

If two very similar terminals exist, one can be defined as being just like the other with certain exceptions. You can specify the string capability `use` with the name of the similar terminal. The capabilities given before `use` override those in the terminal type invoked by `use`. To cancel a capability, place *xx*@ to the left of the capability definition; *xx* is the capability. For example, the following entry defines an AT&T 4424 terminal that does not have the `rev`, `sgr`, and `smul` capabilities, and hence, it cannot do highlighting. This is useful for different modes for a terminal, or for different user preferences. You may specify more than one `use` capability.

```
att4424-2|Teletype 4424 in display function group ii,
                        rev@, sgr@, smul@, use=att4424,
```

FILES

`/usr/lib/tabset/*`	Files that contain tab stop settings for some terminals, in a format appropriate to be output to the terminal (escape sequences that set margins and tab stops)
`/usr/lib/terminfo/?/*`	Files that contain terminal descriptions

SEE ALSO

`term`(5)

`tset`(1B) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

`curses`(3) (available only online)

`printf`(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

`tic`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

`text_tapeconfig` – Tape subsystem configuration file

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The system uses a tape configuration file named `text_tapeconfig` in the `/etc/config` directory. If the `tpinit`(8) command does not find this file, it terminates and returns an error message.

The `text_tapeconfig` file defines all of the tape devices that the system uses. For detailed information on configuring your devices, see the documentation from the tape device vendors.

The diagnostic devices are implicitly defined when the I/O processors (IOPs) and the channels are defined. You may not redefine them.

The tape configuration file consists of comments (optional) and statements. A comment begins with the # symbol and continues to the end of line. A statement consists of a name followed by a list of keyword parameters. There are four statements; two of these statements also consist of substatements. Statements must be in the order shown:

1. `LOADER` statements (one per loader)
2. `DEVICE_GROUP` statements (one per device group)
3. `IOP` statements (one per IOP) or `IONODE` statements (one per node)

   The `IOP` or `IONODE` statement consists of the following two statements that define the IOP or node configuration:

   a. `CHANNEL` statements (one per channel in the IOP or node)

   b. `BANK` statements (one per bank)

      The `BANK` statement consists of two of the following three statements that define the bank configuration:

      i. `SLAVE` statements (one per slave device) (IOS-E only)
      or
      `CONTROL_UNIT` statements (one per control unit)

      ii. `DEVICE` statements (one per device)
4. `OPTIONS` statement

Statement Syntax Rules

The following syntax rules apply to `text_tapeconfig` statements:

- The statement name and its parameters are separated by one or more white spaces (blank, tab, or newline characters).

- Adjacent parameters are separated by a comma.
- The end of the parameter list is indicated by the absence of a comma.
- Adjacent statements are separated by one or more white spaces.

The following is a list of keyword parameter syntax rules:

- The keyword is separated from its value by the `=` symbol.
- The value of a keyword may consist of keywords, numbers, character strings, and lists of keywords, numbers, and character strings.
- If the value of a keyword is a list, the list is enclosed within left and right parentheses. Adjacent elements of a list are separated by a comma. If the list consists of one element, you do not have to enclose it in parentheses. The elements of a list may be lists.
- Numbers may be specified in decimal, octal, and hexadecimal formats. These formats are the same as those used in the C programming language:

Decimal	First digit is not 0 (1372)
Octal	First digit is 0 (0563)
Hexadecimal	First 2 characters are either 0x or 0X (0xf2)

- Character strings are series of characters. If any one of the special characters (white space, `"`, `#`, `=`, `{`, `}`, `(`, `)`, `'`, `\`) is needed in the string, you must enclose the string in a pair of double quotation marks (`"`). Within a pair of double quotation marks, the sequence of characters will be replaced by *x*; *x* is any character. This is the only way you can specify a `"` and a `\` in a quoted string.
- Comments may appear between any symbols described previously.

You can code the names of statements and keywords in a mixture of uppercase and lowercase letters. The values specified by the user is case sensitive. The following specify the same thing:

```
Name = A
name = A
```

The following are different:

```
name = A
name = a
```

The following are descriptions of the tape configuration statements. You must specify a value for each parameter unless a default is specified or the parameter is described as optional.

`LOADER` Statement

The `LOADER` statement identifies the loaders in the `text_tapeconfig` file and has the following format:

`LOADER` *parameter_list*

A description of the parameters follows:

Parameter	Description
`name`	Specifies the loader name, which is the object of several `tpconfig`(8) requests.
`type`	Specifies the loader type. Currently supported types are as follows: `EMASS` EMASS autoloader is used. `IBMTLD` IBM 3494 Tape Library Dataserver is used. `OPERATOR` Operator loads the drive. `STKACS` A StorageTek autoloader that is supported by Cray Research is used.
`status`	Specifies the status (`UP` or `DOWN`) of the loader when the tape daemon starts.
`message_path_to_loader`	Specifies the message path to the servicing loader. `MSGDAEMON` Uses message daemon to send message to loader. `NETWORK` Uses TCP/IP protocol to send message to loader.
`server`	Specifies the server name.
`scratch_volume_label_type`	Specifies the types of scratch requests that the loader may process. If you specify `OPERATOR` for the `type` parameter on the `LOADER` statement, the following types of scratch requests are available. If you specify any other loader type for the `type` parameter only `NONE` is valid. `AL` ANSI labeled scratch tape requests. `NL` Nonlabeled scratch tape requests. `NONE` Scratch labels cannot be used. `SL` IBM standard labeled scratch requests.
`queue_time`	Each volume has a designated "best" loader type for the tape mount. If the best loader is not available, this time is used to queue the tape mount request and to wait for the best loader to become available. If the best loader does not become available during this time, the mount request will be issued to the next best loader. A value of 0 indicates to wait up to 24 hours; a nonzero value specifies the number of seconds to wait.
`verify_non_label_vsn`	Specifies whether the nonlabel VSN should be verified. This parameter may be either `YES` or `NO`.
`message_route_masks`	Routes mount request messages. You can route the mount request message to multiple locations. The list may consist of the following:

`FRONTEND` Issues the mount message to the front end that may be reached through the connection to TPCNET.

`SERVER` Issues the mount message to the server station.

`UNICOS` Issues the mount message to the message daemon. For more information, see `msgdaemon`(8).

`mode` Specifies attended mode:

`ATTENDED` Prompts for operator intervention.

`UNATTENDED` Assumes negative response for operator intervention.

`return_host` Specifies the name of the Cray Research host that serves as the return address for the server.

`child_program_name` *character string*

Specifies the name of the tape daemon loader child program. When the loader is configured `UP`, the child program is activated. This program must be in the directory from which the tape daemon is activated. The directory is typically `/usr/lib/tp`.

If `child_program_name` is omitted, the following values are used:

Loader	**Child Program Name**
EMASS	`esinet`
IBMTLD	`ibmnet`
STKACS	`stknet`

`loader_ring_status`

Specifies whether the loader is alerted to ring status.

`ALERT` Alerts loader to the ring status when a tape is mounted, and checks that the ring status matches the ring status requested by the tape user.

`IGNORE` Ignores the ring status when a tape is mounted. A logical ring out status is used for a tape that has been requested with a ring out status, but its actual ring status is ring in. The default is `ALERT`.

`network_retry_tries` *number*

Specifies the number of times the tape daemon loader child program attempts to send a request over the network to the server after an initial attempt fails. The default for each child program is 5.

`network_send_timeout` *number*

Specifies the time in seconds during which the tape daemon loader child program tries to send a request over the network to the server. The default for each child program is 3 seconds.

`server_reply_wait_time` *number*

Specifies the time in seconds during which a request that is being processed by the server is kept in a queue by the tape daemon loader child program. If a reply has not been received within this time, the child program queries the state of the outstanding request.

The default value for each child program is 180 seconds. For a StorageTek loader, this value is multplied by the number of Library Storage Modules in the Automated Cartridge System.

You must specify at least one `OPERATOR` type loader in the tape configuration file. If the file does not contain such an entity, the tape daemon in its initialization process creates one assuming the following values:

```
LOADER
name = Operator ,
type = OPERATOR ,
status = UP ,
message_PATH_TO_LOADER = MSGDAEMON ,
message_class = NONE ,
server = "" ,
scratch_volume_label_type = NONE ,
queue_time = 1 ,
verify_non_label_vsn = YES ,
message_route_masks = UNICOS ,
mode = ATTENDED ,
return_host = ""
```

The entry created by the tape daemon shows in the output of the `tpmls`(8) command.

DEVICE_GROUP Statement

The tape daemon enforces a resource limit for each user based on the entry for that user in the user database (UDB). The UDB limit applies to resource group numbers, rather than resource or device names. This necessitates a way of mapping the devices configured in the system to the appropriate resource numbers.

The default set of resources (device groups) is determined from the list of devices specified in the `DEVICE_GROUP` statement. The first device group encountered in the list represents resource group number 0. The second device group in the list represents resource group number 1, and so on.

To change this order, specify the `DEVICE_GROUP` statements in the desired order. The first device group name corresponds to resource limit 0. You can list a maximum of eight different device group names.

The mapping specified in this way allows the flexibility of changing the configuration while maintaining a consistent naming convention for device groups that are mapped to the limits set for each user in the UDB.

The `DEVICE_GROUP` statement has the following format:

`DEVICE_GROUP` *parameter_list*

A description of the parameters follows:

Parameter	Description
`name`	Specifies the device group name.
`minlvl`	Specifies the minimum mandatory access control (MAC) level for this device group. The default is the system minimum level 0.
`maxlvl`	Specifies the maximum MAC level for this device group. The default is the system maximum level.
`maxcmp`	Specifies the maximum MAC compartments for this device group. The default is the system maximum compartments.
`avr`	Specifies the status (`YES` or `NO`) of automatic volume recognition (AVR) for this group. The default is the `avr_at_startup` option in the `OPTIONS` statement.
`overcommit`	Specifies whether (`YES` or `NO`) the number of current mount requests can exceed the number of available tape drives. This option overrides the value specified by the `overcommit_at_startup` parameter in the `OPTIONS` statement. If you omit this parameter, the default is the value specified by the `overcommit_at_startup` parameter in the `OPTIONS` statement. For more information about overcommitted mount requests, see the `tpset`(8) and `tpstat`(1) man pages.

`IOP` Statement

The `IOP` statement (IOS-E only) specifies the characteristics of an IOP and has the following format:

`IOP` *parameter_list* `{` *iop_configuration* `}`

iop_configuration consists of a series of `CHANNEL` statements and `BANK` statements following the keyword parameters. Descriptions of the `CHANNEL` and `BANK` statements follow the `IOP` parameters:

Parameter	Description
`number`	Specifies the IOP number. For the CRAY J90 series, this parameter is not used and can be set to 0.
`cluster`	Specifies the cluster number. For the CRAY J90 series, `cluster` is the IOS number.
`type`	Specifies the IOP type (`IOP_BMX`, `IOP_ESCON`, or `IOP_IPI`). The CRAY J90 series do not support this parameter.

`IONODE` Statement

The `IONODE` statement (GigaRing based systems) specifies the characteristics of a node and has the following format:

`IONODE` *parameter_list* `{` *ionode_configuration* `}` *ionode_configuration* consists of a series of `CHANNEL` statements and `BANK` statements following the keyword parameters. Descriptions of the `CHANNEL` and `BANK` statements follow the `IONODE` parameters:

Parameter	**Description**
`node`	Specifies the node number.
`ring`	Specifies the ring number.
`type`	Specifies the node type (`IOP_BMX`, `IOP_ESCON`, or `IOP_MPN`).

`CHANNEL` Statement (`IOP` or `IONODE` Statement)

The `CHANNEL` statement specifies channel characteristics of an IOP or node and has the following format:

`CHANNEL` *parameter_list*

A description of the parameters follows:

Parameter **Description**

`address` Specifies the channel address. The following values are valid:

IOS-E	30, 32, 34, 36
ELS	22-27, 30-37
BMN	0-1
ESN	0-3
MPN	0-7

`microcode_pathname`
(IOS-E only) Specifies the path name of the file that contains the channel microcode and must be specified on the first `CHANNEL` statement of an IOP.

`status` Specifies the status (`UP` or `DOWN`) of the channel when the tape daemon is started.

`adaptor` (IOS-E only) Specifies the channel adapter type (`DCA2`, `FCA2`, or `TCA2`).

`timeout` (IOS-E only) Specifies the ER90 time-out value in seconds. If zero is specified, the IOS-E is set to a time-out period of 10 seconds.

`BANK` Statement (`IOP` or `IONODE` Statement)

The `BANK` statement specifies the bank characteristics of an IOP or node and has the following format:

`BANK` *parameter_list* `{` *bank_configuration* `}`

bank_configuration specifies a series of `SLAVE` or `CONTROL_UNIT` statements followed by a series of `DEVICE` statements. Descriptions of the `SLAVE` and `CONTROL_UNIT` statements follow the optional `BANK` parameter:

Parameter	**Description**
`number`	Specifies a bank number that identifies a bank. Valid values are 0 to 63. Default: a bank number will be assigned to a bank.

SLAVE Statement (BANK Statement)

A SLAVE statement (IOS-E only) specifies the characteristics of a slave device and has the following format:

SLAVE *parameter_list*

A description of the parameters follows:

Parameter	Description
status	Specifies the status (UP or DOWN) of the slave when the tape daemon is started.
path	Specifies a list of channel address and slave address pairs encoded in parentheses. The parentheses are part of the syntax and must be coded. The channel number is the channel that is connected to the port address in the slave.
reset_timeout	Specifies the reset time-out (in seconds).

CONTROL_UNIT Statement (BANK Statement)

The CONTROL_UNIT statement specifies the characteristics of a control unit and has the following format:

CONTROL_UNIT *parameter_list*

A description of the parameters follows:

Parameter **Description**

status
: Specifies the status (UP or DOWN) of the control unit when the tape daemon is started.

protocol
: Specifies the protocol for the control unit, as follows:

SCSI	Specifies the SCSI protocol.
ESCON	Specifies the ESCON protocol. This is the only valid protocol for a control unit attached to an ESCON channel.
INTERLOCK	Specifies interlock protocol.
STREAMING	Specifies data streaming at 3.0 Mbyte/s.
STREAMING45	Specifies data streaming at 4.5 Mbyte/s.

path
: Specifies a list of channel address and control unit port address pairs. For the CRAY J90 series, this channel number is the address specified in the CHANNEL statement. Specifies a list of channel address and control unit port address pairs.

: For SCSI (MPN) tape devices, path specifies a channel and SCSIbus pair. The SCSIbus is determined by looking at the appropriate /opt/CYRIion/adm/mic_code file on the system workstation (SWS). For example, if the file contains the following information, path is 3.

```
sn9132-mpn2, SCSIbus 3 Target 1 Lun [t310], Type = STKSD-3 (VTAPE)
Vendor = STK
Product ID = SD-3
Microcode Rev Level = 0223
Device Min Block Len = 1
Device Max Block Len = 262144
Fixed Block Len = 0(Variable)
Max Block Size = 262144
Default Compression = ON
Ansi Version = 2
Response Format = 2
Attributes = 0x38
```

The channel is restricted to values in the range of 0 through 7 and, by convention, the channel value is the same as the SCSIbus value.

`link_address`
: Specifies the link address of the control unit when it is attached by using an ESCON director. It is set to 0 for directly attached (no director) control units.

`DEVICE` Statement (`BANK` Statement)

The `DEVICE` statement specifies the characteristics of a device and has the following format:

`DEVICE` *parameter_list*

A description of the parameters follows:

Parameter **Description**

`name` Specifies the device name.

`device_group_name`
: Specifies the name of the device group defined by a `DEVICE_GROUP` statement.

`status` Specifies the initial status (`UP` or `DOWN`) of the device.

`id` Specifies the hardware device identifier.

For SCSI (MPN) tape devices, `id` specifies a 3-digit octal number, *xyz*. The following values are valid:

x Is always 0.

y Specifies the SCSI target; that is, the SCSI ID.

z Specifies the tape logical unit (lun).

This information is in the appropriate SWS `/opt/CYRIion/adm/mic_code` file. Although fast and wide devices are supported, SCSI IDs are currently limited to values in the range of 0 through 7.

`type` Specifies the device type.

For SCSI (MPN) tape devices, the value for `type` is in parenthesis following `Type = ` *type* in the appropriate SWS `/opt/CYRIion/adm/mic_code` file. For example, the `type` value for STKSD-3 is `VTAPE`.

`loader` Specifies the loader name defined in a `LOADER` statement.

`vendor_address`
Specifies the vendor address of the drive in an autoloader.

The format for a StorageTek drive is as follows:

acs#,lsm#,panel#,drive#

The format for an EMASS drive is as follows:

drive#

`facility_address`
Specifies only the ER90 facility address.

`short_timeout`
Specifies the ER90 short time-out (in seconds).

`long_timeout`
Specifies the ER90 long time-out (in seconds).

`timeout` Specifies the time-out value in seconds that the ESCON IOP waits for a response from the channel. An integer from 1 to 65535 specifies the number of seconds. A value of 0 directs the tape subsystem to use the time-out value that is hard-coded in the ESCON IOP software. This value is currently set to 900 seconds (15 minutes).

OPTIONS Statement

The options in force when the tape daemon is built are specified in the `/usr/include/tapedef.h` file. You can specify most of these options in the `OPTIONS` section of the `text_tapeconfig` file.

To override the value with which the tape daemon was built, specify the following options and their corresponding values. Descriptions of the options in the `/usr/include/tapedef.h` file are given in the *Tape Subsystem Administration*, Cray Research publication SG–2307. The options that you can specify in the `text_tapeconfig` file with the `OPTIONS` statement are similar to the options in `tapedef.h`, but not identical. Values are often given in a different form in the two files (for example, the value for the `ask_blp` keyword is expressed as 0 or 1 in `tapedef.h`, but it is expressed as `YES` or `NO` in `text_tapeconfig`).

The format of the OPTIONS statement follows:

OPTIONS *parameter_list*

The following parameter list includes valid values or brief definitions of the options.

Parameter	**Description**
allow_unprotected	Allows access (YES or NO) to tapes that do not contain a MAC label in the header. Default: YES
ask_label_switch	Seeks permission (YES or NO) from the operator to switch label type. Default: YES
ask_vsn	Seeks permission (YES or NO) from the operator to specify a VSN when a nonlabel tape is mounted. Default: YES
avr_at_startup	Starts (YES or NO) AVR when the tape daemon is started. Default: YES
blocksize	Specifies the block size to use when the user does not specify a block size by using the tpmnt(1) command -b option. Default: 32768
blp_ring_status	Specifies the user status for the use of the -r option of the tpmnt(1) command when the user requests bypass label processing. UNRESTRICTED specifies the user can use both -r in and -r out. OUT specifies the user can use only -r out. Default: UNRESTRICTED.
check_expiration_date	Specifies whether the operator should check and confirm (YES or NO) the expiration date on the header label of a labeled tape. Default: YES
check_file_id	Specifies whether the file ID on a labeled tape should be checked (YES or NO) when the file is opened. Default: YES
check_protection	Specifies whether the protection flag on the header should be checked (YES or NO). Default: YES
check_vsn	Specifies whether the VSN on labeled tapes should be checked (YES or NO). Default: YES
cray_reel_librarian	Specifies whether the Cray/REELlibrarian system is enabled (YES or NO). Default: NO
cray_reel_librarian_mandatory	Specifies whether the Cray/REELlibrarian system is mandatory (YES or NO). Default: NO
cray_reel_librarian_operator_select_scratch	Indicates whether the operator should verify (YES or NO) the scratch mounts by the Cray/REELlibrarian system before continuing. Default: NO

`cray_reel_librarian_scratch_vsn`
: Specifies the scratch VSN that the Cray/REELlibrarian system will use to tell the operator that a scratch volume is needed. Default: `?CRL??`

`device_group_name`
: Specifies the default device group name if it is not specified on the `-g` option of the `tpmnt`(1) command. Default: `CART`

`file_status`
: Specifies the file status (`NEW` or `OLD`) if it is not specified on the `tpmnt`(1) command. Default: `OLD`

`label_type`
: Specifies the label type (`AL`, `SL`, or `NL`) if it is not specified on the `tpmnt`(1) command. Default: `AL`

`loader_device_assignment_order`
: Specifies the method (`DEVICE_LIST` or `ROUND_ROBIN`) with which the loader assigns devices. Default: `ROUND_ROBIN`

`mainframe_job_origin`
: Specifies the mainframe ID of the job if it is not specified. Default: `C1`

`max_blocksize`
: Sets the upper limit of the block size of the `-b` parameter on the `tpmnt`(1) command. If you specify a larger value, the command terminates abnormally. Default: 4194303

`max_number_of_device_groups`
: Specifies the maximum number of device groups. Default: 8

`max_number_of_tape_users`
: Specifies the maximum number of tape users. Default: 64

`message_daemon_pipename`
: Specifies the message daemon pipe name. Default: `/usr/spool/msg/msg.requests`

`number_of_autoloader_retries`
: Specifies the number of times to try to send a request to the autoloader before informing the operator of an error. The CRAY J90 series do not support this parameter. Default: 10

`operator_message_destination`
: Specifies where operator messages are sent; `UNICOS`, `SERVER`, and `FRONTEND`. Default: `(UNICOS)`

`operator_message_frontend_id`
: Specifies the front-end ID for operator messages. Default: `" "`

`overcommit_at_startup`
: Specifies whether overcommitted mount requests should be enabled as part of start-up when the tape daemon is started (`YES` or `NO`). This option applies only if you omit the `overcommit` parameter on the `DEVICE_GROUP` statement. Default: `NO`

`overcommit_max` Specifies the maximum number of overcommitted mount requests that the tape subsystem can issue. When the number of tape mount requests exceeds this number, the system stops processing requests until one or more of the already overcommitted mount requests are satisfied. You may change this setting by using the `tpset`(8) command. Default: 20

`reselect_cart` Specifies whether another device will be selected (`YES` or `NO`) at end-of-volume for cartridge type devices, which include 3480, 3490, and 3490E devices. Default: `NO`

`retention_period_days`
Specifies the retention period (in days). Default: 0

`ring_status` Specifies the ring status when the ring option (`-r`) is not specified on the `tpmnt`(1) command (`IN`, `OUT`, or `(IN,OUT)`). Default: `(IN,OUT)`

`scratch_volume_action`
Specifies the action (`FREE` or `KEEP`) to perform for scratch tapes when they are released. Default: `FREE`

`scratch_volume_retries`
Specifies the number of retries to get a scratch volume out of the autoloader scratch pool. Default: 3

`scratch_volume_vsn`
Specifies the scratch tape VSN. Default: `??????`

`secure_frontend` Specifies whether security on the front end is enabled (`YES` or `NO`). The CRAY J90 series do not support front-end servicing. Default: `" "`
`servicing_frontend_at_startup` Specifies whether front-end servicing should start (`YES` or `NO`) when the tape daemon is started. The CRAY J90 series do not support any front-end servicing. Default: `NO`

`servicing_frontend_id`
Specifies the servicing front-end ID to use when the `-m` option is missing on the `tpmnt`(1) command. The CRAY J90 series do not support front-end servicing. Default: `" "`

`servicing_frontend_mandatory`
Specifies whether the front-end ID specified by the `servicing_frontend_id` parameter is used (`YES` or `NO`) regardless of the `-m` option on the `tpmnt`(1) command. The CRAY J90 series do not support front-end servicing. Default: `NO`

`servicing_frontend_protocol`
Specifies the protocol (`TCP`) to talk to front ends. The CRAY J90 series do not support front-end servicing. Default: `TCP`

`system_code` Specifies the system code to put on tape labels. Default: `CRI/UNICOS`

`tcp_daemon_childname`
: Specifies the child name of the TCP daemon. The CRAY J90 series do not support this parameter. Default: `tcpnet`

`tcp_daemon_frontend_id`
: Specifies the front-end ID of the TCP daemon. Default: `" "`

`tape_daemon_trace_file_group_id`
: Specifies the group ID of the tape daemon trace files. Default: 9

`tape_daemon_trace_file_mode`
: Specifies the file mode of the tape daemon trace files. Default: 0640

`tape_daemon_trace_file_owner`
: Specifies the owner ID of the tape daemon trace files. Default: 0

`tape_daemon_trace_file_prefix`
: Specifies the tape daemon trace file prefix. Default: `/usr/spool/tape/trace`

`tape_daemon_trace_file_size`
: Specifies the size (in bytes) of the tape daemon trace files. Default: 409600

`tape_daemon_trace_flg`
: Specifies whether tape tracing is enabled (`YES` or `NO`). Default: `YES`

`tape_daemon_trace_savefile_prefix`
: Specifies the prefix to the tape daemon save files. Default: `/usr/spool/tape/trace`

`tcp_daemon_pipename`
: Specifies the pipe name of the TCP daemon. The CRAY J90 series do not support this parameter. Default: `/usr/spool/tape/tcpnet.pipe`

`tcp_daemon_socket_port_number`
: Specifies the socket port number of the TCP daemon. Default: 1167

`user_exit_mask`
: Enables the use of the listed user exits. If no user exits are required, this entry is not needed. For a list of user exits, see the *Tape Subsystem Administration*, Cray Research publication SG–2307. Default: `UEX_NONE`

`verify_scratch_vsn`
: Indicates (`YES` or `NO`) that you may need to send the operator a message that requests verification that a scratch tape is being used to satisfy a tape mount request. You must consult the operator if front-end servicing is not in use. Default: `YES`

EXAMPLES

The following two examples show `text_tapeconfig` files used on IOS-E systems and GigaRing based systems.

For more detail, check the documentation from your tape device vendors; configuration possibilities vary depending up the vendors and the specific devices that you are configuring. For example, an IBM 3490E controller supports multiple devices while a StorageTek Redwood only supports a single device.

GigaRing based systems

The following `text_tapeconfig` file illustrates some typical configurations for GigaRing based systems.

```
LOADER
        name = Operator ,
        type = OPERATOR ,
        status = UP ,
        message_path_to_loader = MSGDAEMON ,
        message_class = NONE ,
        server = UNICOS ,
        scratch_volume_label_type = (NL,AL,SL) ,
        queue_time = 0 ,
        verify_non_label_vsn = NO ,
        message_route_masks = (UNICOS) ,
      loader_ring_status = ALERT,
        mode = ATTENDED
LOADER
        name = stksun ,
        type = STKACS ,
        status = DOWN ,
        message_path_to_loader = NETWORK ,
        message_class = TYPE_340 ,
        server = robot ,
        scratch_volume_label_type = NONE ,
        queue_time = 180 ,
        verify_non_label_vsn = NO ,
        message_route_masks = (UNICOS, FRONTEND) ,
      loader_ring_status = IGNORE,
        mode = ATTENDED

LOADER
        name = wolfy ,
        type = STKACS ,
        status = DOWN ,
        mode = ATTENDED ,
        message_class = TYPE_340 ,
        message_path_to_loader = NETWORK ,
        server = 9490ldr ,
        scratch_volume_label_type = NONE ,
        queue_time = 0 ,
        verify_non_label_vsn = NO ,
```

```
      loader_ring_status = IGNORE,
        message_route_masks = (FRONTEND,UNICOS)

LOADER
        name = panther ,
        type = STKACS ,
        status = DOWN ,
        mode = ATTENDED ,
        message_class = TYPE_340 ,
        message_path_to_loader = NETWORK ,
        server = stk9710 ,
        scratch_volume_label_type = NONE ,
        queue_time = 0 ,
        verify_non_label_vsn = NO ,
      loader_ring_status = IGNORE,
        message_route_masks = (FRONTEND,UNICOS)

LOADER
        name = ibm ,
        type = IBMTLD ,
        status = DOWN ,
        message_path_to_loader = NETWORK ,
        message_class = TYPE_340 ,
        server = ibmtld ,
        scratch_volume_label_type = NONE ,
        queue_time = 15 ,
        verify_non_label_vsn = NO ,
        message_route_masks = UNICOS ,
      loader_ring_status = IGNORE,
        mode = ATTENDED

DEVICE_GROUP
      name = DEFAULT ,
      avr = YES,
      overcommit = NO
DEVICE_GROUP
      name = DAT ,
      avr = YES ,
      overcommit = NO
DEVICE_GROUP
        name = IBM3590 ,
        avr = YES,
      overcommit = NO
DEVICE_GROUP
```

```
          name = IBM3490E ,
          avr = YES ,
        overcommit = NO
  DEVICE_GROUP
          name = STK4890 ,
          avr = YES,
        overcommit = NO
  DEVICE_GROUP
        name = DLT4000 ,
        avr = YES,
        overcommit = NO
  DEVICE_GROUP
          name = STK9490 ,
          avr = YES,
        overcommit = NO
  DEVICE_GROUP
        name = STKSD3 ,
        avr = YES,
        overcommit = NO
  IONODE
        node     = 1 ,
        ring     = 0 ,
        type     = IOP_MPN
        {
              CHANNEL
                    address = 0 ,
                    status  = up
              CHANNEL
                    address = 1 ,
                    status  = up
              CHANNEL
                    address = 2 ,
                    status  = up
              CHANNEL
                    address = 3 ,
                    status  = up
              CHANNEL
                    address = 4 ,
                    status  = up
              CHANNEL
                    address = 5 ,
                    status  = up
              CHANNEL
                    address = 6 ,
```

```
      status  = up
CHANNEL
      address = 7 ,
      status  = up
    BANK
            number = 1
            {
                    CONTROL_UNIT
                            status = UP ,
                            path = (1,1) ,
                            protocol = SCSI
                    DEVICE
                            name = s4890s0,
                  device_group_name = STK4890,
                            id = 000 ,
                            type = 3490E ,
                            status = DOWN ,
                  vendor_address = (0,0,2,0) ,
                            loader = panther
                    DEVICE
                            name = s4890s1,
                    device_group_name = STK4890,
                            id = 010 ,
                            type = 3490E ,
                            status = DOWN ,
                        vendor_address = (0,0,2,1) ,
                            loader = panther
                    DEVICE
                            name = d4000s0,
                   device_group_name = DLT4000,
                            id = 020 ,
                            type = VTAPE ,
                            status = DOWN ,
                   vendor_address = (0,0,2,2) ,
                             loader = panther
                     DEVICE
                             name = d4000s1,
                     device_group_name = DLT4000,
                             id = 030 ,
                             type = VTAPE ,
                             status = DOWN ,
                   vendor_address = (0,0,2,3) ,
                             loader = panther
      }
```

```
                BANK
                        number = 6
                        {
                                CONTROL_UNIT
                                        status = UP ,
                                        path = (6,6),
                                        protocol = SCSI
                                DEVICE
                                        name =  3490s0,
                                        device_group_name = IBM3490E ,
                                        id = 060,
                                        type = 3490E,
                                        status = DOWN ,
                                        loader = ibm
                                DEVICE
                                        name =  3490s1,
                                        device_group_name = IBM3490E ,
                                        id = 061,
                                        type = 3490E,
                                        status = DOWN ,
                                        loader = ibm
                 }
      }
IONODE
      node     = 1 ,
      ring     = 1 ,
      type     = IOP_MPN
      {
            CHANNEL
                  address = 0 ,
                  status  = up
            CHANNEL
                  address = 1 ,
                  status  = up
            CHANNEL
                  address = 2 ,
                  status  = up
            CHANNEL
                  address = 3 ,
                  status  = up
            CHANNEL
                  address = 4 ,
                  status  = up
            CHANNEL
```

```
            address = 5 ,
            status  = up
      CHANNEL
            address = 6 ,
            status  = up
      CHANNEL
            address = 7 ,
            status  = up
      BANK
            number = 17
            {
                  CONTROL_UNIT
                        status    = up ,
                        path      = (7,7) ,
                        protocol  = SCSI
                          DEVICE
                                  name =  d5649JX,
                                  device_group_name = DAT ,
                                  id = 040 ,
                                  type = VTAPE,
                                  status = DOWN ,
                                  loader = Operator
            }
      BANK
            number = 16
            {
                  CONTROL_UNIT
                        status    = up ,
                        path      = (6,6) ,
                        protocol  = SCSI
                          DEVICE
                                  name =  s9490s0,
                                  device_group_name = STK9490 ,
                                  id = 000 ,
                                  type = 3490E,
                                  status = down ,
                                  vendor_address = (0,0,1,0) ,
                                  loader = wolfy
                          DEVICE
                                  name =  s9490s1,
                                  device_group_name = STK9490 ,
                                  id = 010 ,
                                  type = 3490E,
                                  status = DOWN ,
```

```
                                vendor_address = (0,0,1,1) ,
                                loader = wolfy
                        DEVICE
                                name =  s9490s2,
                                device_group_name = STK9490 ,
                                id = 020 ,
                                type = 3490E,
                                status = DOWN ,
                                vendor_address = (0,0,1,2) ,
                                loader = wolfy
                        DEVICE
                                name =  s9490s3,
                                device_group_name = STK9490 ,
                                id = 030 ,
                                type = 3490E,
                                status = DOWN ,
                                vendor_address = (0,0,1,3) ,
                                loader = wolfy
         }
        BANK
               number = 10
               {
                        CONTROL_UNIT
                                status = UP ,
                                path = (0, 0),
                                protocol = SCSI
                        DEVICE
                                name =  3590s0,
                                device_group_name = IBM3590 ,
                                id = 000,
                                type = VTAPE,
                                status = DOWN ,
                                loader = ibm
         }
        BANK
               number = 12
               {
                        CONTROL_UNIT
                                status = UP ,
                                path = (2, 2),
                                protocol = SCSI
                        DEVICE
                                name =  3590s1,
                                device_group_name = IBM3590 ,
```

```
                                    id = 010,
                                    type = VTAPE,
                                    status = DOWN ,
                                    loader = ibm
                  }
                 BANK
                        number = 11
                        {
                         CONTROL_UNIT
                               status    = up ,
                               path      = (1,1) ,
                               protocol  = SCSI
                                 DEVICE
                                        name =  ssd3_s0,
                                        device_group_name = STKSD3 ,
                                        id = 010 ,
                                        type = VTAPE,
                                        status = DOWN ,
                               vendor_address = (0,0,3,1) ,
                                       loader = wolfy
                  }
                 BANK
                        number = 13
                        {
                         CONTROL_UNIT
                               status    = up ,
                               path      = (3,3) ,
                               protocol  = SCSI
                                 DEVICE
                                        name =  ssd3_s1,
                                        device_group_name = STKSD3 ,
                                        id = 030 ,
                                        type = VTAPE,
                                        status = DOWN ,
                               vendor_address = (0,0,3,3) ,
                                        loader = wolfy
                  }
         }

        OPTIONS
        allow_unprotected             = YES ,
        ask_label_switch              = YES ,
        ask_vsn                             = NO ,
        avr_at_startup                      = YES ,
```

```
blp_ring_status                    = UNRESTRICTED ,
blocksize                    = 65536 ,
check_expiration_date              = YES ,
check_file_id                      = YES ,
check_protection             = NO ,
check_vsn                    = YES ,
cray_reel_librarian                = NO ,
cray_reel_librarian_mandatory      = NO ,
cray_reel_librarian_operator_select_scratch    = NO ,
cray_reel_librarian_scratch_vsn          = ?CRL?? ,
device_group_name            = DEFAULT ,
file_status                  = OLD ,
label_type                   = AL ,
loader_device_assignment_order           = ROUND_ROBIN ,
mainframe_job_origin               = C1 ,
max_number_of_device_groups        = 8 ,
max_blocksize                      = 4194303 ,
max_number_of_tape_users           = 100 ,
message_daemon_pipename            = /usr/spool/msg/msg.requests ,
number_of_autoloader_retries       = 10 ,
operator_message_destination       = UNICOS ,
operator_message_frontend_id       = "" ,
operator_message_type              = USCP_TYPE_1 ,
overcommit_at_startup              = NO,
overcommit_max                     = 20,
reselect_cart                      = NO ,
retention_period_days              = 0 ,
ring_status                  = (IN,OUT) ,
scratch_volume_action              = FREE ,
scratch_volume_retries             = 3 ,
scratch_volume_vsn                 = ?????? ,
secure_frontend              = NO ,
servicing_frontend_at_startup      = NO ,
servicing_frontend_id              = "" ,
servicing_frontend_mandatory       = NO ,
servicing_frontend_protocol        = TCP ,
stop_hippi_eiop                    = YES ,
system_code                        = CRI/UNICOS ,
tape_daemon_trace_file_group_id    = 9 ,
tape_daemon_trace_file_mode        = 0640 ,
tape_daemon_trace_file_owner       = 0 ,
tape_daemon_trace_file_prefix      = /usr/spool/tape/trace ,
tape_daemon_trace_file_size        = 409600 ,
tape_daemon_trace_flg                   = YES ,
```

```
    tape_daemon_trace_savefile_prefix   = /usr/spool/tape/save/trace ,
    tcp_daemon_childname                = tcpnet ,
    tcp_daemon_frontend_id              = "mvs" ,
    tcp_daemon_pipename                 = /usr/spool/tape/tcpnet.pipe ,
    tcp_daemon_socket_port_number       = 1167 ,
    user_exit_mask                      = (UEX_NONE) ,
    verify_scratch_vsn                  = YES
```

IOS-E support

The following `text_tapeconfig` file illustrates some typical configurations for systems with IOS-E support.

```
LOADER
        name = Operator ,
        type = OPERATOR ,
        status = UP ,
        mode = ATTENDED ,
        message_path_to_loader = MSGDAEMON ,
        server = UNICOS ,
        scratch_volume_label_type = (NL,AL,SL) ,
        queue_time = 0 ,
        verify_non_label_vsn = YES ,
        message_route_masks = (UNICOS,FRONTEND)

LOADER
        name = stksun ,
        type = STKACS ,
        status = DOWN ,
        mode = ATTENDED ,
        message_path_to_loader = NETWORK ,
        server = robot ,
        scratch_volume_label_type = NONE ,
        queue_time = 15 ,
        verify_non_label_vsn = NO ,
        message_route_masks = (UNICOS)

LOADER
        name = esisun ,
        type = EMASS ,
        status = DOWN ,
        mode = ATTENDED ,
        message_path_to_loader = NETWORK ,
        server = er90-sun ,
        scratch_volume_label_type = NONE ,
        queue_time = 15 ,
```

```
        verify_non_label_vsn = NO ,
        message_route_masks = (UNICOS)

DEVICE_GROUP
        name = CART

DEVICE_GROUP
        name = TAPE

DEVICE_GROUP
        name = SILO

DEVICE_GROUP
        name = 3490

DEVICE_GROUP
        name = 3490E ,
        avr  = YES

DEVICE_GROUP
        name = TEST

DEVICE_GROUP
        name = ER90

DEVICE_GROUP
        name = ESCON

IOP
        number   = 3 ,
        cluster  = 0 ,
        type     = IOP_IPI
        {
                CHANNEL
                        address = 036 ,
                        status  = UP ,
                        microcode_pathname = /etc/micro/IPI3.ucode ,
                        timeout = 10000 ,
                        adaptor = DCA2
                CHANNEL
                        address = 034 ,
                        status  = UP ,
                        microcode_pathname = /etc/micro/IPI3.ucode ,
                        timeout = 10000 ,
```

```
                adaptor = DCA2
        BANK
                number = 1
                {
                        SLAVE
                                path = (034,0) ,
                                status = UP ,
                                reset_timeout = 1000
                        DEVICE
                                name   = er92 ,
                                device_group_name = ER90 ,
                                id     = 0 ,
                                type   = ER90 ,
                                status = DOWN ,
                                loader = esisun ,
                                vendor_address = 2 ,
                                facility_address = 0xFF ,
                                short_timeout = 600 ,
                                long_timeout  = 400
                }
        BANK
                number = 2
                {
                        SLAVE
                                path = (036,0) ,
                                status = UP ,
                                reset_timeout = 1000
                        DEVICE
                                name   = er93 ,
                                device_group_name = ER90 ,
                                id     = 0 ,
                                type   = ER90 ,
                                status = DOWN ,
                                loader = esisun ,
                                vendor_address = 3 ,
                                facility_address = 0xFF ,
                                short_timeout = 600 ,
                                long_timeout  = 400
                }
    }
IOP
        number   = 1 ,
        cluster  = 0 ,
        type     = IOP_BMX
```

```
{
        CHANNEL
                address = 030 ,
                status  = UP ,
                microcode_pathname = /etc/micro/TCA1.ucode
        CHANNEL
                address = 036 ,
                status  = UP ,
                microcode_pathname = /etc/micro/TCA1.ucode
        CHANNEL
                address = 034 ,
                status  = UP ,
                microcode_pathname = /etc/micro/TCA1.ucode
        CHANNEL
                address = 032 ,
                status  = UP ,
                microcode_pathname = /etc/micro/TCA1.ucode
        BANK
                number = 3
                {
                        CONTROL_UNIT
                                status    = UP ,
                                path      = ((036, 0)) ,
                                protocol  = INTERLOCK
                        DEVICE
                                name   = 220 ,
                                device_group_name = TAPE ,
                                id     = 00 ,
                                type   = 3420 ,
                                status = DOWN ,
                                loader = Operator
                        DEVICE
                                name   = 221 ,
                                device_group_name = TAPE ,
                                id     = 01 ,
                                type   = 3420 ,
                                status = DOWN ,
                                loader = Operator
                }

        BANK
            number = 4
                {
                        CONTROL_UNIT
```

```
                                        status    = UP ,
                                        path      = ((034,11),(030,11),(032,0)) ,
                                        protocol  = STREAMING45
                                DEVICE
                                        name   = 120 ,
                                        device_group_name = CART ,
                                        id     = 00 ,
                                        type   = 3480 ,
                                        status = DOWN ,
                                        loader = Operator
                                DEVICE
                                        name   = 300 ,
                                        device_group_name = CART ,
                                        id     = 00 ,
                                        type   = 3480 ,
                                        status = DOWN ,
                                        loader = stksun ,
                                        vendor_address = (0,0,10,0)
                                DEVICE
                                        name   = 301 ,
                                        device_group_name = CART ,
                                        id     = 01 ,
                                        type   = 3480 ,
                                        status = DOWN ,
                                        loader = stksun ,
                                        vendor_address = (0,0,10,1)
                                DEVICE
                                        name   = 170 ,
                                        device_group_name = 3490E ,
                                        id     = 04 ,
                                        type   = 3490E ,
                                        status = DOWN ,
                                        loader = Operator
                  }
      }
IOP
        number   = 3 ,
        cluster  = 1 ,
        type     = IOP_ESCON
        {
                CHANNEL
                        address = 036 ,
                        status  = UP ,
                        microcode_pathname = /etc/micro/FCA2.ucode ,
```

```
                adaptor = FCA2
        CHANNEL
                address = 034,
                status  = UP ,
                microcode_pathname = /etc/micro/FCA2.ucode,
                adaptor = FCA2
         BANK
                number = 5
{
                        CONTROL_UNIT
                                status    = UP ,
                                path      = ((036, 00)) ,
                                protocol  = ESCON ,
                                link_address = 0

                        DEVICE
                                name   = 170e ,
                                device_group_name = ESCON ,
                                id     = 00 ,
                                type   = 3490E ,
                                status = DOWN ,
                                loader = operator
}

        BANK
                number = 6
{
                        CONTROL_UNIT
                                status    = UP ,
                                path      = ((034, 00)) ,
                                protocol  = ESCON ,
                                link_address = C3

                        DEVICE
                                name   = 34C300 ,
                                device_group_name = ESCON ,
                                id     = 00 ,
                                type   = 3490E ,
                                status = DOWN ,
                                loader = operator

                        DEVICE
                                name   = 34C301 ,
                                device_group_name = ESCON ,
```

```
                                id     = 01 ,
                                type   = 3490E ,
                                status = DOWN ,
                                loader = operator
        }
                BANK
                        number = 7
        {
                                CONTROL_UNIT
                                        status    = UP ,
                                        path = ((034, 00)),
                                        protocol = ESCON ,
                                        link_address = C8

                                DEVICE
                                        name   = 34C800 ,
                                        device_group_name = ESCON ,
                                        id     = 00 ,
                                        type   = 3490E ,
                                        status = DOWN ,
                                        loader = operator
        }
                BANK
                        number = 8
        {
                                CONTROL_UNIT
                                        status    = UP ,
                                        path      = ((030, 00)),
                                        protocol  = ESCON ,
                                        link_address = C3

                                CONTROL_UNIT
                                        path      = ((032, 00)) ,
                                        status    = UP ,
                                        protocol  = ESCON ,
                                        link_address = C3

                                DEVICE
                                        name   = device1 ,
                                        device_group_name = ESCON ,
                                        id     = 00 ,
                                        type   = 3490E ,
                                        status = DOWN ,
                                        loader = operator
```

```
        }
OPTIONS
allow_unprotected                       = YES,
ask_label_switch                        = YES ,
ask_vsn                                 = YES ,
avr_at_startup                          = YES ,
blp_ring_status                         = UNRESTRICTED ,
blocksize                               = 32768 ,
check_expiration_date                   = YES ,
check_file_id                           = YES ,
check_protection                        = YES ,
check_vsn                               = YES ,
cray_reel_librarian                     = NO ,
cray_reel_librarian_mandatory           = NO ,
cray_reel_librarian_operator_select_scratch     = NO ,
cray_reel_librarian_scratch_vsn         = ?CRL?? ,
device_group_name                       = CART ,
file_status                             = OLD ,
label_type                              = AL ,
loader_device_assignment_order          = ROUND_ROBIN ,
mainframe_job_origin                    = C1 ,
max_number_of_device_groups             = 8 ,
max_blocksize                           = 4194303 ,
max_number_of_tape_users                = 64 ,
message_daemon_pipename                 = /usr/spool/msg/msg.requests ,
number_of_autoloader_retries            = 10 ,
operator_message_destination            = (UNICOS) ,
operator_message_frontend_id            = "" ,
overcommit_at_startup                   = NO ,
overcommit_max                          = 20 ,
reselect_cart                           = NO ,
retention_period_days                   = 0 ,
ring_status                             = (IN,OUT) ,
scratch_volume_action                   = FREE ,
scratch_volume_retries                  = 3 ,
scratch_volume_vsn                      = ?????? ,
secure_frontend                         = NO ,
servicing_frontend_at_startup           = NO ,
servicing_frontend_id                   = "" ,
servicing_frontend_mandatory            = NO ,
servicing_frontend_protocol             = TCP ,
system_code                             = CRI/UNICOS ,
tape_daemon_trace_file_group_id         = 9 ,
tape_daemon_trace_file_mode             = 0640 ,
```

```
tape_daemon_trace_file_owner            = 0 ,
tape_daemon_trace_file_prefix           = /usr/spool/tape/trace ,
tape_daemon_trace_file_size             = 409600 ,
tape_daemon_trace_flg                   = YES ,
tape_daemon_trace_savefile_prefix       = /usr/spool/tape/save/trace ,
tcp_daemon_childname                    = tcpnet ,
tcp_daemon_frontend_id                  = " " ,
tcp_daemon_pipename                     = /usr/spool/tape/tcpnet.pipe ,
tcp_daemon_socket_port_number           = 1167 ,
user_exit_mask                          = UEX_NONE ,
verify_scratch_vsn                      = YES ,
```

FILES

`/etc/config/text_tapeconfig`	Tape subsystem configuration file
`/usr/include/tapedef.h`	Definitions for trace file size
`/usr/include/tapereq.h`	Tape daemon interface definition file

SEE ALSO

`msgdaemon`(8), `tpconf`(8), `tpconfig`(8), `tpinit`(8), `tpmls`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

Tape Subsystem Administration, Cray Research publication SG–2307

NAME

tmpdir.users – List of authorized users for tmpdir(1)

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The /etc/tmpdir.users file contains a list of users and their authorizations for the tmpdir(1) command.

The tmpdir.users file is an ASCII file that contains one entry for each authorized user. Entries have the following format:

user_name : *path_name* [: *path_name*]

The first field in each line is the user name (login name). The remaining fields specify the path names of directories in which the user may create temporary directories. The fields are separated by colons; each entry is separated from the next by a new-line character.

FILES

/etc/tmpdir.users	List of authorized users for tmpdir(1)

SEE ALSO

tmpdir(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

NAME

`types` – Definition of primitive system data types

SYNOPSIS

```
#include <sys/types.h>
```

IMPLEMENTATION

Cray PVP systems

DESCRIPTION

The data types used in UNICOS system code are defined in the `sys/types.h` include file. Some data of these types is accessible to user code.

```
typedef long           word;
typedef unsigned long  ulong;
typedef unsigned int   uint;
typedef unsigned short ushort;
typedef long           blkno_t;
typedef long           daddr_t;
typedef struct inode   inode_t;
typedef word           label_t[128];  /* save area for Bs,Ts */

typedef word *         waddr_t;
typedef unsigned char  uchar;
typedef short          cnt_t;
typedef long           paddr_t;
typedef long           key_t;
```

The `daddr_t` format is used for disk addresses except in an inode on disk; see `fs`(5). Times are encoded in seconds since 00:00:00 GMT, January 1, 1970. The major and minor parts of a device code specify the kind and unit number of a device and are installation-dependent. Offsets are measured in bytes from the beginning of a file. The `label_t` array of variables is used to save the processor state while another process is running.

FILES

`/usr/include/sys/types.h` Data types definition file

SEE ALSO

`fs`(5)

NAME

udb – Format of the user database (UDB) file

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The user database (UDB) files contain control information for users of the UNICOS operating system and for the fair-share scheduler's resource groups. The UDB files replace the /etc/passwd file as the primary source for user validation and control information. The UDB consists of the following files:

- /etc/udb
- /etc/udb.public
- /etc/udb_2/udb.index
- /etc/udb_2/udb.priva
- /etc/udb_2/udb.pubva

The files in the /etc/udb_2 directory extend the capability of the UDB beyond what was available in previous releases.

To allow users access to nonsensitive UDB information, the /etc/udb.public, /etc/udb_2/index, and /etc/udb_2/udb.pubva files are publicly readable. The other files contain privileged information, such as encrypted passwords and security information, and only privileged callers can read them. Write access to all files is restricted to privileged users.

You should write these files only through the supplied library routines described in libudb(3C). Because of the reliance on file locking to maintain the integrity of the files, other access methods may corrupt the database and require regeneration of the files.

Organization of /etc/udb

The /etc/udb file is organized in blocks of 4096 characters (one sector) to provide the ability to ensure atomic updates of information. The file is organized in blocks, as follows:

Block	**Description**
0	File header for validation, version control, and default information
1-4	User ID (UID) hash blocks
5-8	Name hash blocks
9-*n*	User information blocks

The user information blocks contain a bidirectional chain that links records in numeric order by UID. This provides the mechanism that implements the next UID and previous UID access functions.

When new records are added to the file, the first empty record slot is allocated and the new record is written to that position. Then the linkage is adjusted to place the new record in the correct logical place in the user ID space. If no empty slot exists, the file will be extended by one block and the new record added at the end of the file. The released library configuration specifies three user records per block.

Organization of extension files

The extension UDB files exist in the `/etc/udb_2` directory. The three extension files are `udb.index`, `udb.priva`, and `udb.pubva`. The format of these files is described in this subsection, following a description of the common file header. You can find the formal declarations of these files in the file `libc/udb/libudb.h`.

Common file header

Each file has a 4096-byte header that contains control information used by the access method. Important information in the header data includes the magic number, which identifies the file, and the version identifier, which identifies the structure of the file. The header contains space for the default UDB table (`struct udbdefault`) and the tape name table (`struct udbtmap`), but this space is used only in `udb.pubva`.

Index file `udb.index`

The `udb.index` index file consists of a common file header, an index header, and two index arrays. The first index array is the name array, and the second is the UID array. The size of the index file depends on the number of records in the database.

The index arrays are packed together following the index header; the entire file occupies one or more 4096-byte blocks with possible free space at the end. The `length` field in the index header reflects this length. The `entries` field in the index header specifies the number of entries actually in use in both arrays. (Each array contains the same number of entries.)

The name array is sorted in ascending order on the `name` field, as determined by `strcmp`(3C); duplicate entries are not allowed. The UID array is sorted on ascending value of UID; duplicates are allowed but the order is arbitrary.

In each index entry, the `pub_pos` field specifies the disk offset of the target record in the public file; the `priv_pos` field is the offset of the start of the record in the private file.

Data files `udb.priva` and `udb.pubva`

The `udb.priva` and `udb.pubva` data files have the same format. The `udb.pubva` file contains public information, and `udb.priva` contains private information. A data file begins with a common file header, followed by an arbitrary number of free records and data records.

Free records are chained together and linked to the free-chain pointer in the common data header. Free records are used, if possible, when new records are created or when a record expands and must be moved to find sufficient contiguous space.

A data record begins with a header that contains the following information:

- Name and UID of the record
- Length and compression information
- Time the record was last changed

- Other control information
- Compressed data fields

The header is designed to provide sufficient information to reconstruct a damaged database without having to decompress the data.

The compressed data is a variable-sized extension to the record header that has been compressed into a bit stream to reduce its size and to remove unnecessary fields. All zero-length fields are deleted, because the decompression process restores their 0 value in the `udb` structure without needing any recorded information to do so. Each nonzero data field consists of an identifying token, a length, and a value. Compressed data can be copied but cannot be decompressed without the decompression algorithm in the access method library.

Format of User Entry

The format of a user entry as defined in the `/usr/include/udb.h` file is a property of `libudb`, and the interface is described in `libudb`(3C).

NOTES

You can find the external representation of a record in the user database in the `libudb.3c` file. To save space in the file, much of the information is packed as densely as is practical, using the structure defined in the library source file (`libc/gen/uentrydb.c`). Transformation functions within the library convert between external and internal representation for the caller.

FILES

`/etc/udb`	User information
`/etc/udb.public`	Public user information
`/etc/udb_2/udb.index`	Public extension index file
`/etc/udb_2/udb.priva`	Private field extension file
`/etc/udb_2/udb.pubva`	Public field extension file
`/usr/include/sharedefs.h`	File of reasons for eviction by the fair-share scheduler
`/usr/include/sys/secparm.h`	File of user permissions
`/usr/include/udb.h`	Structure definition of user database files

SEE ALSO

`acid`(5), `group`(5), `passwd`(5)

`udbsee`(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

`libudb`(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

`udbgen`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

updaters – Configuration file for NIS updating

SYNOPSIS

/etc/yp/updaters

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The /etc/yp/updaters file is a makefile (see make(1)) that updates network information services (NIS) databases. You can update databases only in a secure network; that is, a network that has a publickey(5) database. Each entry in the file is a make target for a particular NIS database. For example, if an NIS database named passwd.byname can be updated, a make target named passwd.byname should be in the updaters file with the appropriate command to update the file.

The requesting client passes the information necessary to make the update to the ypupdated(8) program, which passes the information to the update(3X) command through standard input. This information is described in the following list:

- Network name of client wanting to make the update (a string)
- Type of update (an integer)
- Number of bytes in key (an integer)
- Actual bytes of key
- Number of bytes in data (an integer)
- Actual bytes of data

Each of the items is followed by a newline character, except for actual bytes of key and actual bytes of data.

After getting this information through standard input, the command to update the particular database decides whether the user is allowed to make the change. If not, the command exits with the status of NISERR_ACCESS. If the user is allowed to make the change, the command makes the change and exits with a status of 0. If any errors exist that can prevent the updater from making the change, updaters exits with the status that matches a valid NIS error code described in the rpcsvc/ypclnt.h file.

FILES

/etc/yp/updaters	File that updates NIS databases

SEE ALSO

`publickey`(5)

`make`(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011,

`ypupdated`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

utmp, wtmp – utmp and wtmp file formats

SYNOPSIS

```
#include <sys/types.h>
#include <utmp.h>
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The utmp and wtmp files hold user information for such commands as login(1), who(1), and write(1). Accounting programs such as csaline(8) and acctcon1(8) also process the utmp and wtmp files. These files have the following structure, as defined by the utmp.h include file:

```
#define UTMP_FILE  "/etc/utmp"
#define WTMP_FILE  "/etc/wtmp"
#define ut_name     ut_user

struct utmp {
       char    ut_user[8];         /* User login name              */
       char    ut_id[4];           /* /etc/lines ID (usually line #)*/
       char    ut_line[12];        /* device name (console, lnxx)   */
       char    ut_host[24];        /* Name of remote machine        */
       short   ut_pid;             /* process ID                    */
       short   ut_type;            /* type of entry                 */
       struct  exit_status  {
               short  e_termination; /* Process termination status  */
               short  e_exit;        /* Process exit status         */
       } ut_exit;                    /* Exit status of process      */
                                     /* marked as DEAD_PROCESS      */
       time_t  ut_time;              /* time entry was made         */
       char    ut_tpath[TPATHSIZ];   /* path of temporary file      */
       short   ut_jid;               /* job ID of pgrp leader       */
} ;
```

```
/*     Definitions for ut_type */
#define  EMPTY         0
#define  RUN_LVL       1
#define  BOOT_TIME     2
#define  OLD_TIME      3
#define  NEW_TIME      4
#define  INIT_PROCESS  5         /* Process spawned by "init"       */
#define  LOGIN_PROCESS 6         /* A process waiting for login     */
#define  USER_PROCESS  7         /* A user process                  */
#define  DEAD_PROCESS  8
#define  ACCOUNTING    9
#define  UTMAXTYPE  ACCOUNTING   /* Largest legal value of ut_type  */

/*  Special strings or formats used in the "ut_line" field  */
/*  when accounting for something other than a process      */
/*  No string for the ut_line field can be more than        */
/*  11 chars + a NULL in length                             */
#define  RUNLVL_MSG  "run-level %c"
#define  BOOT_MSG    "system boot"
#define  OTIME_MSG   "old time"
#define  NTIME_MSG   "new time"
```

FILES

/etc/utmp	File of user information
/etc/wtmp	File of user information
/usr/include/sys/types.h	Data type definition file
/usr/include/utmp.h	Format definition for the utmp and wtmp files

SEE ALSO

last(1B), login(1), who(1), write(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

getut(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

acctcon1(8), csaline(8), fwtmp(8), wtmpfix(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

uuencode – Encoded uuencode file format

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

Files output by uuencode(1) consist of a header line, followed by a number of body lines, and a trailer line. The uudecode(1) command ignores any lines that precede the header or following the trailer. Lines preceding a header must not look like a header.

The header line is distinguished by having begin the first six characters. The word begin is followed by a mode (in octal) and a string that names the remote file. A space separates the three items in the header line.

The body consists of a number of lines, each consisting of at most 62 characters (including the trailing newline character). These consist of a character count, followed by encoded characters, followed by a newline character. The character count is one printing character, and it represents an integer, the number of bytes the rest of the line represents. Such integers are always in the range from 0 to 63 and can be determined by subtracting the character space (octal 40) from the character.

Groups of 3 bytes are stored in 4 characters, 6 bits per character. All are offset by a space to make the characters printing. The last line may be shorter than the normal 45 bytes. If the size is not a multiple of 3, this fact can be determined by the value of the count on the last line. Extra characters will be included to make the character count a multiple of 4. The body is terminated by a line with a count of 0. This line consists of one ASCII space.

The trailer line consists of end on a line by itself.

SEE ALSO

mail(1) for electronic message system information
uudecode(1) to decode a binary file
in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

NAME

`values` – Machine-dependent values definition file

SYNOPSIS

```
#include <values.h>
```

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `values.h` include file contains a set of manifest constants defined for Cray Research processor architectures. The model assumed for integers is binary representation (twos complement), where the sign is represented by the value of the high-order bit.

The most important constants are defined as follows:

`DSIGNIF` Number of significant bits in the mantissa of a double-precision, floating-point number

`FSIGNIF` Number of significant bits in the mantissa of a single-precision, floating-point number

`HIBITI` Value of a regular integer with only the high-order bit set (usually the same as `HIBITS` or `HIBITL`)

`HIBITL` Value of a long integer with only the high-order bit set

`HIBITS` Value of a short integer with only the high-order bit set

`MAXDOUBLE`, `LN_MAXDOUBLE`
Maximum value of a double-precision, floating-point number and its natural logarithm

`MAXFLOAT`, `LN_MAXFLOAT`
Maximum value of a single-precision, floating-point number and its natural logarithm

`MAXINT` Maximum value of a signed regular integer (usually the same as `MAXSHORT` or `MAXLONG`)

`MAXLONG` Maximum value of a signed long integer

`MAXSHORT` Maximum value of a signed short integer

`MINDOUBLE`, `LN_MINDOUBLE`
Minimum positive value of a double-precision, floating-point number and its natural logarithm

`MINFLOAT`, `LN_MINFLOAT`
Minimum positive value of a single-precision, floating-point number and its natural logarithm

FILES

`/usr/include/values.h` Machine-dependent values definitions

SEE ALSO

`float.h`(3C), `numeric_lim`(3C), `values.h`(3C) in the *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

NAME

ypfiles – Network information service (NIS) database and directory structure

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The network information service (NIS) function provides a simple network look-up service that consists of databases and processes. The NIS network look-up service uses a database of dbm files in the /etc/yp directory.

A dbm database consists of two files created by calls to the dbm(3C) library package. One has the file name extension .pag and the other has the file name extension .dir. For instance, the database named hosts.byname is implemented by the pair of files names hosts.byname.pag and hosts.byname.dir.

A dbm database served by the NIS is called an NIS *map*. An NIS *domain* is a named set of NIS maps. Each NIS domain is implemented as a subdirectory of /etc/yp. Any number of NIS domains can exist; each may contain any number of maps.

The NIS look-up service itself requires no maps, although they may be required for the normal operation of other parts of the system. There is no list of maps that NIS serves; if the map exists in a given domain and a client asks about it, the NIS serves it. For a map to be accessible consistently, it must exist on all NIS servers for the domain. To provide data consistency between the replicated maps, make an entry to transfer the NIS map periodically from each NIS server (with the ypxfr(8) command) /usr/lib/crontab on each server.

NIS maps should contain key-value pairs that consist of the YP_LAST_MODIFIED key and the YP_MASTER_NAME key. YP_LAST_MODIFIED is the order number or time (in seconds) when the map was built; its value is a 10-character ASCII number. YP_MASTER_NAME is the name of the NIS master server. The makedbm(8) command generates the key-value pairs automatically. NIS can serve a map that does not contain key-value pairs, but the ypserv(8) process cannot return values for a Get_order_number or Get_master_name request. When ypxfr(8) transfers a map from a master NIS server to a slave, ypxfr(8) also uses the values of these two keys.

You must generate and modify NIS maps only at the master server. To copy them to the slaves, use ypxfr(8). This prevents potential byte-ordering problems among NIS servers running on machines that have different architectures and reduces the amount of disk space required for the dbm files. To set up the NIS database initially for both masters and slaves, use ypinit(8).

After the server databases are set up, the contents of some maps probably will change. Generally, an ASCII source version of the database exists on the master. To change this version, use a text editor. The edited copy is incorporated into the NIS map and is propagated from the master to the slaves by running the `/etc/yp/yp.mk` makefile. All standard maps have entries in `/etc/yp/yp.mk`; if you add an NIS map, edit this file to support the new map. The makefile uses `makedbm`(8) to generate the NIS map on the master and `yppush`(8) to propagate the changed map to the slaves. The `yppush`(8) command is a client of the map `ypservers`, which lists all the NIS servers.

NOTES

The NIS was formerly known as yellow pages, which explains the `yp`-prefix on command and directory names.

SEE ALSO

`makedbm`(8), `rpcinfo`(8), `ypinit`(8), `yppoll`(8), `yppush`(8), `ypserv`(8), `ypxfr`(8) in the *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

NAME

`intro` – Miscellaneous information pages

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

Section 7D contains miscellaneous documentation, mostly concerning DWB. DWB, which is based on AT&T's Documenter's Workbench, runs under UNICOS and provides a variety of macro packages.

In addition to DWB man pages, this section includes a man page for `msg`(7D), the text formatting macros for UNICOS messages.

NAME

`eqnchar` – Special character definitions for `eqn`(1)

SYNOPSIS

`eqn /usr/pub/eqnchar` [*filename*] | `troff` [*options*]

`neqn /usr/pub/eqnchar` [*filename*] | `nroff` [*options*]

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `eqnchar` command contains `troff`(1) and `nroff`(1) character definitions for constructing characters that are not available on the Graphic Systems typesetter. These definitions are primarily intended for use with `eqn`(1) and `neqn`(1). It contains definitions for the following characters:

<table>
<tr><td>ciplus</td><td>⊕</td><td>||</td><td>‖</td><td>square</td><td>□</td></tr>
<tr><td>citimes</td><td>⊗</td><td>langle</td><td>〈</td><td>circle</td><td>○</td></tr>
<tr><td>wig</td><td>~</td><td>rangle</td><td>〉</td><td>blot</td><td>■</td></tr>
<tr><td>-wig</td><td>≃</td><td>hbar</td><td>ℏ</td><td>bullet</td><td>•</td></tr>
<tr><td>>wig</td><td>≳</td><td>ppd</td><td>⊥</td><td>prop</td><td>∝</td></tr>
<tr><td><wig</td><td>≲</td><td><-></td><td>↔</td><td>empty</td><td>∅</td></tr>
<tr><td>=wig</td><td></td><td><=></td><td></td><td>member</td><td>∈</td></tr>
<tr><td>star</td><td>∗</td><td>|<</td><td>≮</td><td>nomem</td><td>∉</td></tr>
<tr><td>bigstar</td><td>∗</td><td>|></td><td>≯</td><td>cup</td><td>∪</td></tr>
<tr><td>=dot</td><td>≐</td><td>ang</td><td>∠</td><td>cap</td><td>∩</td></tr>
<tr><td>orsign</td><td>∨</td><td>rang</td><td>∟</td><td>incl</td><td>⊑</td></tr>
<tr><td>andsign</td><td>∧</td><td>3dot</td><td>⋮</td><td>subset</td><td>⊂</td></tr>
<tr><td>=del</td><td>≜</td><td>thf</td><td>∴</td><td>supset</td><td>⊃</td></tr>
<tr><td>oppA</td><td>∀</td><td>quarter</td><td>¼</td><td>!subset</td><td>⊆</td></tr>
<tr><td>oppE</td><td>∃</td><td>3quarter</td><td>¾</td><td>!supset</td><td>⊇</td></tr>
<tr><td>angstrom</td><td>Å</td><td>degree</td><td>°</td><td></td><td></td></tr>
</table>

FILES

`/usr/pub/eqnchar`

SEE ALSO

`eqn`(1), `nroff`(1), `troff`(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

NAME

man – Macros to format AT&T reference manual pages

SYNOPSIS

nroff -man *filename* ...

troff -man *filename* ...

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

These macros are used to lay out the reference pages in this manual. If *filename* contains format input for a preprocessor, the preceding commands must be piped through the appropriate preprocessor. man(1) handles this automatically. See the Conventions subsection.

Any text argument *t* may be 0 to 6 words. You may use quotation marks to include SPACE characters in a word. If *text* is empty, the special treatment is applied to the next input line with text to be printed. In this way, you may use .I to italicize a whole line, or use .SB to make small bold letters.

A prevailing indent distance is remembered between successive indented paragraphs, and it is reset to default value on reaching a nonindented paragraph. Default units for indents *i* are ens.

Type font and size are reset to default values before each paragraph, and after processing font and size setting macros.

These strings are predefined by -man:

*R ‘ ’, ‘(Reg)’ in nroff

*S Change to default type size.

Requests

n.t.l. Next text line

p.i. Prevailing indent

Request	Cause break	If no argument	Explanation
.B *t*	No	*t*=n.t.l.	Text is in bold font.
.BI *t*	No	*t*=n.t.l.	Join words, alternating bold and italic.
.BR *t*	No	*t*=n.t.l.	Join words, alternating bold and roman.
.DT	No	.5i 1i...	Restore default tabs.

Request	Cause break	If no argument	Explanation
`.HP` *i*	Yes	*i*=p.i.	Begin paragraph with hanging indent. Set prevailing indent to *i*.
`.I` *t*	No	*t*=n.t.l.	Text is italic.
`.IB` *t*	No	*t*=n.t.l.	Join words, alternating italic and bold.
`.IP` *x i*	Yes	*x*=""	Same as `.TP` with tag *x*.
`.IR` *t*	No	*t*=n.t.l.	Join words, alternating italic and roman.
`.IX` *t*	No	–	Index macro, for Sun internal use.
`.LP`	Yes	–	Begin left-aligned paragraph. Set prevailing indent to 0.5i.
`.PD` *d*	No	*d*=.4v	Set vertical distance between paragraphs.
`.PP`	Yes	–	Same as `.LP`.
`.RE`	Yes	–	End of relative indent. Restore prevailing indent.
`.RB` *t*	No	*t*=n.t.l.	Join words, alternating roman and bold.
`.RI` *t*	No	*t*=n.t.l.	Join words, alternating roman and italic.
`.RS` *i*	Yes	*i*=p.i.	Start relative indent, increase indent by *i*. Set prevailing indent to 0.5i for nested indents.
`.SB` *t*	No	–	Reduce size of text by 1 point and make text bold.
`.SH` *t*	Yes	–	Section heading.
`.SM` *t*	No	*t*=n.t.l.	Reduce size of text by 1 point.
`.SS` *t*	Yes	*t*=n.t.l.	Section subheading.
`.TH` *n s d f m*	Yes	–	Begin reference page *n*, of section *s*; *d* is the date of the most recent change. If present, *f* is the left page footer; *m* is the main page (center) header. Set prevailing indent and tabs to 0.5i.
`.TP` *i*	Yes	*i*=p.i.	Begin indented paragraph, with the tag given on the next text line. Set prevailing indent to *i*.
`.TX` *t p*	No	–	Resolve the title abbreviation *t*; join to punctuation mark (or text) *p*.

Conventions

When formatting a man page, `man`(1) examines the first line to determine whether it requires special processing. For example, a first line consisting of the following code indicates that the man page must be run through the `tbl`(1) preprocessor:

```
'\" t
```

A typical manual page for a command or function is laid out as follows:

`.TH` *title* [1–8]

The name of the command or function, which serves as the title of the manual page. This is followed by the number of the section in which it appears.

`.SH NAME`

The name, or list of names, by which the command is called, followed by a dash and then a one-line summary of the action performed. All in roman font, this section contains no `troff`(1) commands or escapes, and no macro requests. It is used to generate the `whatis`(1) database.

`.SH SYNOPSIS`

Commands — The syntax of the command and its arguments, as typed on the command line. When in bold, you must type a word exactly as printed. When in italics, you can replace a word with an argument. References to bold or italicized items are not capitalized in other sections, even when they begin a sentence.

Syntactic symbols appear in roman face:

- `[ ]` An argument, when surrounded by brackets, is optional.
- `|` Arguments separated by a vertical bar are exclusive. You can supply only one item from such a list.
- ... Arguments followed by an ellipsis can be repeated. When an ellipsis follows a bracketed set, you can repeat the expression within the brackets.

Functions — If required, the data declaration, or `#include` directive, is shown first, followed by the function declaration. Otherwise, the function declaration is shown.

`.SH DESCRIPTION`

A narrative overview of the command or function's external behavior. This includes how it interacts with files or data, and how it handles the standard input, standard output, and standard error. Internals and implementation details are usually omitted. This section tries to provide a succinct overview.

Literal text from the synopsis appears in constant width, as do literal file names and references to items that appear elsewhere in the reference manuals.

Arguments are italicized. If a command interprets either subcommands or an input grammar, its command interface or input grammar is usually described in a USAGE section, which follows the OPTIONS section. The DESCRIPTION section describes only the behavior of the command itself, not that of subcommands.

`.SH OPTIONS`

The list of options, along with a description of how each affects the command's operation.

`.SH FILES`

A list of files associated with the command or function.

`.SH SEE ALSO`
: A comma-separated list of related man pages, followed by references to other published materials.

`.SH DIAGNOSTICS`
: A list of diagnostic messages and an explanation of each.

`.SH BUGS`
: A description of limitations, known defects, and possible problems associated with the command or function.

FILES

`/usr/lib/tmac/tmac.an`

SEE ALSO

`man`(1), `nroff`(1), `tbl`(1), `troff`(1), `whatis`(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

NAME

`me` – Macros for formatting papers

SYNOPSIS

`nroff -me` [*options*] *filename* ...

`troff -me` [*options*] *filename* ...

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `me` package of `nroff`(1) and `troff`(1) macro definitions provides a canned formatting facility for technical papers in various formats. When producing two-column output on a terminal, filter the output through `col`(1).

A definition of the macro requests follow. Many `nroff`(1) and `troff`(1) requests are unsafe in conjunction with this package; however, you may use these requests with impunity after the first `.pp`:

`.bp` Begins new page.

`.br` Breaks output line here.

`.sp` *n* Inserts *n* spacing lines.

`.ls` *n* (line spacing) *n*=1 single; *n*=2 double-space.

`.na` Does not align right margin.

`.ce` *n* Centers next *n* lines.

`.ul` *n* Underlines next *n* lines.

`.sz` +*n* Adds *n* to point size.

Output of the `eqn`(1), `neqn`(1), and `tbl`(1) preprocessors for equations and tables is acceptable as input.

Requests

In the following list, *initialization* refers to the first `.pp`, `.lp`, `.ip`, `.np`, `.sh`, or `.uh` macro. This list is incomplete.

Request	Initial value	Cause break	Explanation
`.(c`	–	Yes	Begins centered block.
`.(d`	–	No	Begins delayed text.
`.(f`	–	No	Begins footnote.

Request	Initial value	Cause break	Explanation
`.(l`	–	Yes	Begins list.
`.(q`	–	Yes	Begins major quote.
`.(x`*x*	–	No	Begins indexed item in index *x*.
`.(z`	–	No	Begins floating keep.
`.)c`	–	Yes	Ends centered block.
`.)d`	–	Yes	Ends delayed text.
`.)f`	–	Yes	Ends footnote.
`.)l`	–	Yes	Ends list.
`.)q`	–	Yes	Ends major quote.
`.)x`	–	Yes	Ends index item.
`.)z`	–	Yes	Ends floating keep.
`.++` *m H*	–	No	Defines paper section. *m* defines the part of the paper, and it can be `C` (chapter), `A` (appendix), `P` (preliminary, for instance, abstract, table of contents, and so on), `B` (bibliography), `RC` (chapters renumbered from page one each chapter), or `RA` (appendix renumbered from page 1).
`.+c` *T*	–	Yes	Begins chapter (or appendix, and so on, as set by `.++`). *T* is the chapter title.
`.1c`	1	Yes	One-column format on a new page.
`.2c`	1	Yes	Two-column format.
`.EN`	–	Yes	Space after equation produced by `eqn`(1) or `meqn`.
`.EQ` *x y*	–	Yes	Precedes equation; break out and add space. Equation number is *y*. The optional argument *x* may be *I* to indent equation (default), *L* to left-adjust the equation, or *C* to center the equation.
`.GE`	–	Yes	Ends *gremlin* picture.
`.GS`	–	Yes	Begins *gremlin* picture.
`.PE`	–	Yes	Ends `pic`(1) picture.
`.PS`	–	Yes	Begins `pic`(1) picture.
`.TE`	–	Yes	Ends table.

Request	Initial value	Cause break	Explanation
`.TH`	–	Yes	Ends heading section of table.
`.TS` *x*	–	Yes	Begins table; if *x* is *H*, table has repeated heading.
`.ac` *A N*	–	No	Sets up for ACM style output. *A* is the Author's name(s), and *N* is the total number of pages. Must be given before the first initialization.
`.b` *x*	No	No	Prints *x* in bold face; if no argument, switches to bold face.
`.ba` +*n*	0	Yes	Augments the base indent by *n*. Use this indent to set the indent on regular text (such as paragraphs).
`.bc`	No	Yes	Begins new column.
`.bi` *x*	No	No	Prints *x* in bold italics (no-fill only).
`.bu`	–	Yes	Begins bulleted paragraph.
`.bx` *x*	No	No	Prints *x* in a box (no-fill only).
`.ef` *′x′y′z*	′′′′′	No	Sets even footer to *x y z*.
`.eh` *′x′y′z*	′′′′′	No	Sets even header to *x y z*.
`.fo` *′x′y′z*	′′′′′	No	Sets footer to *x y z*.
`.hx`	–	No	Suppresses headers and footers on next page.
`.he` *′x′y′z*	′′′′′	No	Sets header to *x y z*.
`.hl`	–	Yes	Draws a horizontal line.
`.i` *x*	No	No	Italicizes *x*; if *x* missing, italic text follows.
`.ip` *x y*	No	Yes	Starts indented paragraph, with hanging tag *x*. Indentation is *y* ens (default 5).
`.lp`	Yes	Yes	Starts left-blocked paragraph.
`.lo`	–	No	Reads in a file of local macros of the form `.*`*x*. Must be given before initialization.
`.np`	1	Yes	Starts numbered paragraph.
`.of` *′x′y′z*	′′′′′	No	Sets odd footer to *x y z*.
`.oh` *′x′y′z*	′′′′′	No	Sets odd header to *x y z*.
`.pd`	–	Yes	Prints delayed text.
`.pp`	No	Yes	Begins paragraph. First line is indented.

Request	Initial value	Cause break	Explanation
`.r`	Yes	No	Roman text follows.
`.re`	–	No	Resets tabs to default values.
`.sc`	No	No	Reads in a file of special characters and diacritical marks. Must be given before initialization.
`.sh` *n x*	–	Yes	Section head follows, font automatically bold. *n* is level of section, and *x* is title of section.
`.sk`	No	No	Leaves the next page blank. Only one page is remembered ahead.
`.sm` *x*	–	No	Sets *x* in a smaller point size.
`.sz` +*n*	10p	No	Augments the point size by *n* points.
`.th`	No	No	Produces the paper in thesis format. Must be given before initialization.
`.tp`	No	Yes	Begins title page.
`.u` *x*	–	No	Underlines argument (even in `troff`(1)). (No-fill only).
`.uh`	–	Yes	Like `.sh`, but unnumbered.
`.xp` *x*	–	No	Prints index *x*.

FILES

`/usr/lib/tmac/*.me`

`/usr/lib/tmac/e`

SEE ALSO

`col`(1), `eqn`(1), `nroff`(1), `pic`(1), `tbl`(1), `troff`(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

NAME

`ms` – Text formatting macros

SYNOPSIS

`nroff -ms` [*options*] *filename* ...

`troff -ms` [*options*] *filename* ...

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The `ms` package of `nroff`(1) and `troff`(1) macro definitions provides a formatting facility for various styles of articles, theses, and books. When producing two-column output on a terminal or line printer, or when reverse-line motions are needed, filter the output through `col`(1). Definitions of all external `-ms` macros follow.

NOTE: This `-ms` macro package is an extended version written at Berkeley and is a superset of the standard `-ms` macro packages as supplied by Bell Labs. Some of the Bell Labs macros have been removed; for instance, it is assumed that the user has little interest in producing headers stating that the memo was generated at Whippany Labs.

Many `nroff`(1) and `troff`(1) requests are unsafe in conjunction with this package. However, you may use the first four requests that follow with impunity after initialization, and you may use the last two even before initialization:

`.bp`	Begins new page.
`.br`	Breaks output line.
`.sp` *n*	Inserts *n* spacing lines.
`.ce` *n*	Centers next *n* lines.
`.ls` *n*	Line spacing: *n*=1 single; *n*=2 double-space.
`.na`	Does not align right margin.

Font and point size changes with `\f` and `\s` also are allowed (for example, `\fI`*word*`\fR` italicizes *word*). Output of the `tbl`(1) and `eqn`(1) preprocessors for equations and tables is acceptable as input.

Requests

Macro name	Initial value	Break? Reset?	Explanation
`.AB` x	–	y	Begins abstract; if *x* = no, do not label abstract.
`.AE`	–	y	Ends abstract.
`.AI`	–	y	Author's institution.
`.AM`	–	n	Better accent mark definitions.
`.AU`	–	y	Author's name.
`.B` *x*	–	n	Emboldens *x*; if no *x*, switches to bold face.
`.B1`	–	y	Begins text to be enclosed in a box.
`.B2`	–	y	Ends boxed text and prints it.
`.BT`	date	n	Bottom title, printed at foot of page.
`.BX` *x*	–	n	Prints word *x* in a box.
`.CM`	if t	n	Cuts mark between pages.
`.CT`	–	y,y	Chapter title: page number moved to CF (TM only).
`.DA` *x*	if n	n	Forces date *x* at bottom of page; today if no *x*.
`.DE`	–	y	Ends display (unfilled text) of any kind.
`.DS` *Y*	I	y	Begins display with keep; *x* = I, L, C, B; *y* = indent.
`.ID`*z*	8n,.5i	y	Indented display with no keep; *y* = indent.
`.LD`	–	y	Left display with no keep.
`.CD`	–	y	Centered display with no keep.
`.BD`	–	y	Block display; centers entire block.
`.EF` *x*	–	n	Even page footer *x* (three part as for `.tl`).
`.EH` *x*	–	n	Even page header *x* (three part as for `.tl`).
`.EN`	–	y	Ends displayed equation produced by `eqn`(1).
`.EQ` *Y*	–	y	Breaks out equation; *x* = L,I,C; *y* = equation number.
`.FE`	–	n	Ends footnote to be placed at bottom of page.
`.FP`	–	n	Numbered footnote paragraph; may be redefined.
`.FS` *x*	–	n	Starts footnote; *x* is optional footnote label.
`.HD`	undef	n	Optional page header below header margin.
`.I` *x*	–	n	Italicizes *x*; if no *x*, switches to italics.
`.IP` *Y*	–	y,y	Indented paragraph, with hanging tag *x*; *y* = indent.
`.IX` *Y*	–	y	Indexes words *x y* and so on (up to five levels).
`.KE`	–	n	Ends keep of any kind.
`.KF`	–	n	Begins floating keep; text fills remainder of page.
`.KS`	–	y	Begins keep; unit kept together on a single page.

Macro name	Initial value	Break? Reset?	Explanation
`.LG`	–	n	Larger; increases point size by 2.
`.LP`	–	y,y	Left (block) paragraph.
`.MC` *x*	–	y,y	Multiple columns; *x* = column width.
`.ND` *x*	if t	n	No date in page footer; *x* is date on covers.
`.NH` *Y*	–	y,y	Numbered header; *x* = level, *x* = *0* resets, *x* = S sets to *y*.
`.NL`	10p	n	Sets point size back to normal.
`.OF` *x*	–	n	Odd page footer *x* (three part as for `.tl`).
`.OH` *x*	–	n	Odd page header *x* (three part as for `.tl`).
`.P1`	if TM	n	Prints header on first page.
`.PP`	–	y,y	Paragraph with first line indented.
`.PT`	- % -	n	Page title, printed at head of page.
`.PX` *x*	–	y	Prints index (table of contents); *x* = no suppresses title.
`.QP`	–	y,y	Quotes paragraph (indented and shorter).
`.R`	on	n	Returns to roman font.
`.RE`	5n	y,y	Retreats: ends level of relative indentation.
`.RP` *x*	–	n	Released paper format; *x* = no stops title on first page.
`.RS`	5n	y,y	Right shifts: starts level of relative indentation.
`.SH`	–	y,y	Section header, in bold face.
`.SM`	–	n	Smaller; decreases point size by 2.
`.TA`	8n,5n	n	Sets TAB characters to 8n 16n ... (`nroff`(1)) 5n 10n ... (`troff`(1)).
`.TC` *x*	–	y	Prints table of contents at end; *x* = no suppresses title.
`.TE`	–	y	Ends table processed by `tbl`(1).
`.TH`	–	y	Ends multipage header of table.
`.TL`	–	y	Title in bold face and two points larger.
`.TM`	off	n	UC Berkeley thesis mode.
`.TS` *x*	–	y,y	Begins table; if *x* = H, table has multipage header.
`.UL` *x*	–	n	Underlines *x*, even in `troff`(1).
`.UX` *x*	–	n	UNIX; trademark message first time; *x* appended
`.XA` *Y*	–	y	Another index entry; *x* = page or no for none; *y* = indent.
`.XE`	–	y	Ends index entry (or series of `.IX` entries).
`.XP`	–	y,y	Paragraph with first line exdented, others indented.
`.XS` *Y*	–	y	Begins index entry; *x* = page or no for none; *y* = indent.
`.1C`	on	y,y	One-column format, on a new page.

Macro name	Initial value	Break? Reset?	Explanation
`.2C`	–	y,y	Begins two-column format.
`.]-`	–	n	Beginning of `refer` reference.
`.[0`	–	n	Ends unclassifiable type of reference.
`.[N`	–	n	N = 1:journal-article, 2:book, 3:book-article, 4:report.

Registers

To control formatting distances in `-ms`, use built-in number registers. For example, the following command line sets the line length to 6.5 inches:

```
.nr  LL  6.5i
```

A table of number registers and their default values follows:

Name	Register controls	Takes effect	Default
`PS`	Point size	Paragraph	10
`VS`	Vertical spacing	Paragraph	12
`LL`	Line length	Paragraph	6i
`LT`	Title length	Next page	Same as `LL`
`FL`	Footnote length	Next `.FS`	5.5i
`PD`	Paragraph distance	Paragraph	1v (if n), 0.3v (if t)
`DD`	Display distance	Displays	1v (if n), 0.5v (if t)
`PI`	Paragraph indent	Paragraph	5*n*
`QI`	Quote indent	Next `.QP`	5*n*
`FI`	Footnote indent	Next `.FS`	2*n*
`PO`	Page offset	Next page	0 (if n), ~1i (if t)
`HM`	Header margin	Next page	1i
`FM`	Footer margin	Next page	1i
`FF`	Footnote format	Next `.FS`	0 (1, 2, 3 available)

When resetting these values, make sure to specify the appropriate units. Setting the line length to 7, for example, results in output with one character per line; setting `FF` to 1 suppresses footnote superscripting; setting it to 2 also suppresses indentation of the first line; and setting it to 3 produces an `.IP`-like footnote paragraph.

A list of string registers available in `-ms` follows; you may use them anywhere in the text:

Name	String's Function
`\*Q`	Quote (" in `nroff`(1), ‘‘ in `troff`(1))
`\*U`	Unquote (" in `nroff`(1), ’’ in `troff`(1))
`\*-`	Dash (`--` in `nroff`(1), — in `troff`(1))
`\*(MO`	Month (month of the year)
`\*(DY`	Day (current date)
`\**`	Automatically numbered footnote
`\*´`	Acute accent (before letter)
`` \*` ``	Grave accent (before letter)
`\*^`	Circumflex (before letter)
`\*,`	Cedilla (before letter)
`\*:`	Umlaut (before letter)
`\*~`	Tilde (before letter)

When using the extended accent mark definitions available with `.AM`, these strings should come after, rather than before, the letter to be accented.

BUGS

Floating keeps and regular keeps are diverted to the same space; therefore, you cannot mix them together with predictable results.

FILES

`/usr/lib/tmac/ms.???`

`/usr/lib/tmac/s`

SEE ALSO

`col`(1), `eqn`(1), `nroff`(1), `tbl`(1), `troff`(1) in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

NAME

msg – Text formatting macros for UNICOS messages

SYNOPSIS

nroff -msg *files*
troff -msg *files*

IMPLEMENTATION

All Cray Research systems

DESCRIPTION

The msg macros are a package of nroff and troff macro definitions that provides a formatting facility for the printed documents of the UNICOS message system. The Requests subsection defines all available macros.

Virtually all nroff and troff directives should be unnecessary in conjunction with this macro package. However, they are available if desired and should work as documented. You also can create tables and equations by using tbl(1) and eqn(1) directives, respectively; these processors work with the msg macros.

For a description of how to format and print a file that uses the msg macros, see the *Cray Message System Programmer's Guide*, Cray Research publication SG–2121.

Requests

Unless otherwise noted, all msg requests (macros) must start at the beginning of a line. No other text lines or words can start with a . symbol.

.2S Starts two-column mode. This macro automatically makes point size equal to 9 and vertical spacing (leading) equal to 11.

.2E Ends two-column mode.

.BL [*x*] (Bulleted list) Makes an entry in a bulleted list. The *x* is either d for double-spaced list or s for single-spaced list (the default).

.CF "*string*" (Center footer string) Defines *string* as the center string for footers. In Cray Research style, the center footer contains the security level of the manual (public, private, or proprietary).

.CH *x* "*string1*" "*string2*"
(Column headings) Makes underlined column heads for two-column lists; *x* is indent (as in _TL); *string1* and *string2* are column heads. *x* cannot be less than 1.28 or greater than 47. If *x* is less than the width of *string1*, the width of *string1* is used as the first column width. If the first column width would leave the second column less than 5-ens wide, *x* is adjusted to keep the second column 5-ens wide.

`.CR` [*x*] [*y*] (Counter reset) Resets the counter that is output by the `\*n` string. The *x* argument is the number at which the next count will start (default is 1). The *y* argument is the type of the counter character (default is Arabic numerals). For a list of the values that you may use for *y*, see the description of the *x* argument of the `.NL` macro).

`.CS` (Code start) Begins a block of code in Courier font. In two-column mode (point size 9), lines between `.CS` and `.CE` macros cannot consist of more than 44 characters.

`.CE` (Code end) Ends a code block started with `.CS`.

`.DL` [*x*] (Dashed list) Makes an entry in a dashed list. The *x* is either `d` for double-spaced list or `s` for single-spaced list (the default).

`.EQ` Starts equation (with `eqn`); ends equation with `.EN`.

`.EN` Ends equation (with `eqn`).

`.GC` "*group*"

(Group code) Defines the group code to be used for a set of messages. You must define the group code because it is printed as part of the message identifier for each message.

`.KT` *x*[`i`] (Keep together) Keeps the next *x* output lines or the next *x* inches from breaking over a page. *x* is interpreted as a line count unless you specify the `i` suffix. In that case, it is interpreted as a number of inches (`3.9i`, for example). If *x* is more than 53, the `_KT` macro is ignored.

`.MN` "*string*" (Manual number) Defines the manual (publication) number for page footers as *string*.

`.MS` *msg#* [`b`]

This macro is substituted automatically for `$nexp` tags by the `catxt`(1) command.

(Message start) Starts a message block that, by default, will not break over columns or pages, unless it is longer than one column (page). To force a message to break over a column or page, use the `b` option. The *msg-#* is the message number used in the online message system. If the text will be used in the UNICOS message system, this argument is required.

`.ME` (Message end) Ends a message block.

`.MT` "*string*" (Manual title) Defines the manual title for use in page headers as *string*.

`.NL` [*x* [*y* [*z* [`d`]]]]

(Numbered list) Makes an entry in a numbered list. The *x* is either the type of numerals you want (default is Arabic), or a `d` to specify a double-spaced, Arabic-numbered list. The *y* is the indent between the numerals and the paragraph. The *z* is the number at which to (re)start the count if you want something other than the first character in the series (1 or i or A, and so on). If you want a double-spaced list that uses something other than Arabic numerals (so that you cannot specify `d` for *x*), specify the `d` as a fourth argument to `.NL`. The *x* argument can have one of the following values:

Value	Default indent	Result
`1`	3.3	Arabic numerals (the default)
`a`	3.3	Lowercase letters
`A`	4	Uppercase letters
`i`	4.5	Lowercase roman numerals
`I`	5	Uppercase roman numerals
`d`	3.3	Arabic numerals, with full blank lines between list entries

You should end numbered lists with the `.NN` macro.

`.NN` (Numbered-list end) Ends numbered list (resets numbers to 1 at that level of indent).

`.PP` [*x*] (Paragraph (resets indent)) *x* is the number of (printed) lines to keep together on the same page; the default is 4. Use this argument only if you use the `b` option to `.MS` or if your message is longer than one column.

`.RN` [*fig-no* [*pg-no* [*tbl-no* [*sec-no* [*sec-style*]]]]]
(Renumber) Placed at the head of a section you want to print by itself (without preceding sections), this macro starts figure, page, table, and section numbers at the values specified. The last argument is either 0 for numeric section numbers (the default) or `A` for alphabetic section numbers (used in appendixes).

`.SN` (Sequential numbering) Ensures that pages, figures, and tables are numbered sequentially across multiple files; specify as the last line of each file. Also, allows multiple sections in the same file; put just before any `.ST` macros other than the very first one in a file.

`.SP` [*x*] Adds vertical space (leading) without resetting indentation. Use `_SP` instead of `_PP` in indented lists and examples). The *x* is the number of following lines to keep in one block (not break over pages).

`.SQ` [*x*] Space half a line (use instead of `_PP` in indented lists and examples). The *x* is the number of following lines to keep in one block (not break over pages).

`.ST` "*string*" (Section title) *string* is the section title.

`.TL` [*x* [*y*]] (Tagged list) Makes a tagged-paragraph list. The following line is the tag, and, on only the first entry, *x* is the indent; it cannot be less than 1.28 or greater than 47, and if you do not specify it, it defaults to 5 ens. The *y* is either `d` for double-spaced list or `s` for single-spaced list (the default).

`.TS` (Table start) Begins a table to be processed with `tbl`. Be sure that tables are 3.3 i. or narrower in width.

`.TE` (Table end) Ends a table to be processed by `tbl`.

Font Changes

`\fB`	Change to New Century bold font (can start anywhere on a line).
`\fI` or `\*V`	Change to New Century italic font (can start anywhere on a line).
`\fR`	Change to New Century roman font (can start anywhere on a line).
`\*C`	Change to Courier font (can start anywhere on a line).
`\*(Cb` or `\f(CB`	Change to Courier bold font (can start anywhere on a line).
`\*(Co` or `\f(CO`	Change to Courier italic font (can start anywhere on a line).
`\fP`	Change to previous font (use to undo font change).

Nonprinting Comments

`´\"`	Comment line; entire line is ignored when formatted (this version is preferred).

Predefined Strings

Hardware Names

`\*y`	`\%CRAY\ Y-MP`; the resulting text, "CRAY Y-MP", will not break over lines.
`\*(ys`	`\%CRAY\ Y-MP\ EL`; the resulting text, "CRAY Y-MP EL", will not break over lines.
`\*(EL`	`\%EL\ series`; the resulting text, "EL series", will not break over lines.
`\*(EO`	`\%EL\ IOS`; the resulting text, "EL IOS", will not break over lines.
`\*(c9`	`\%CRAY\ C90`; the resulting text, "CRAY C90", will not break over lines.
`\*(Ie`	`\%IOS-E`; the resulting text, "IOS-E", will not break over lines.
`\*(IE`	`\%IOS model\ E`; the resulting text, "IOS model E", will not break over lines.
`\*m`	`\%CRAY\ T3D`; the resulting text, "CRAY T3D", will not break over lines.
`\*(MP`	`\%Cray\ MPP\ systems`; the resulting text, "Cray MPP systems", will not break over lines

Miscellaneous:

`\*(Ca`	CRI
`\*(Cr`	Cray Research, Inc.
`\*(UM`	`\%UNICOS\ MAX`; the resulting text, "UNICOS MAX", will not break over lines
`\*u`	UNICOS

FILES

`/usr/lib/tmac/tmac.sg` Message macro package

SEE ALSO

`catxt`(1), `explain`(1) describe UNICOS message system user commands
`eqn`(1), `nroff`(1), `tbl`(1), `troff`(1) describe text formatting utilities
in the *UNICOS User Commands Reference Manual*, Cray Research publication SR–2011

Cray Message System Programmer's Guide, Cray Research publication SG–2121, contains details about all aspects of the message system