Tape Subsystem Administration

SG–2307 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.

New Features

Tape Subsystem Administration SG–2307 10.0

With the UNICOS 10.0 release, this manual documents the tape subsystem for both the UNICOS and UNICOS/mk operating systems. Its title, *UNICOS Tape Subsystem Administrator's Guide*, has been changed to *Tape Subsystem Administration*, Cray Research publication SG–2307. Any administration differences between the operating systems are highlighted in the text.

This revision documents the new and updated user exits, additional `errupt`(8) information, updated configuration information, and man page changes.

Miscellaneous editorial and technical changes were made.

Record of Revision

Version	*Description*
9.0	August 1995 Original Printing.
9.2	December 1996 Online documentation supports the administration of the UNICOS 9.2 release running on the Cray Research computer systems.
9.3	August 1997 Online documentation supports the administration of the UNICOS 9.3 release running on the Cray Research computer systems.
10.0	November 1997 Online documentation supports the administration of the UNICOS 10.0 release running on the Cray Research computer systems and subsequent UNICOS/mk running on Cray T3E systems.

Contents

Page

Figures

Tables

Preface

This manual documents UNICOS 10.0 release running on Cray Research systems. It contains information needed in the administration of the tape subsystem available to UNICOS and UNICOS/mk systems.

Warning: Starting with the UNICOS 10.0 release, the term *Cray ML-Safe* replaces the term *Trusted UNICOS*, which referred to the system configuration used to achieve the UNICOS 8.0.2 release evaluation. Because of changes to available software, hardware, and system configurations since the UNICOS 8.0.2 system release, the term *Cray ML-Safe* does not imply an evaluated product, but refers to the currently available system configuration that closely resembles that of the evaluated Trusted UNICOS 8.0.2 system.

For the UNICOS 10.0 release, the functionality of the Trusted UNICOS system has been retained, but the `CONFIG_TRUSTED` option, which enforces conformance to the strict B1 configuration, is no longer available.

UNICOS system administration publications

Information on the structure and operation of a Cray Research computer system running the UNICOS operating system, as well as information on administering various products that run under the UNICOS operating system, is contained in the following documents:

- *General UNICOS System Administration*, Cray Research publication SG–2301, contains information on performing basic administration tasks as well as information about system and security administration using the UNICOS multilevel (MLS) feature. This publication contains chapters documenting file system planning, UNICOS startup and shutdown procedures, file system maintenance, basic administration tools, crash and dump analysis, the UNICOS multilevel security (MLS) feature, and administration of online features.
- *UNICOS Resource Administration*, Cray Research publication SG–2302, contains information on the administration of various UNICOS features available to all UNICOS systems. This publication contains chapters documenting accounting, automatic incident reporting (AIR), the fair-share scheduler, file system quotas, file system monitoring, system activity and performance monitoring, and the Unified Resource Manager (URM).

- *UNICOS Configuration Administrator's Guide*, Cray Research publication SG–2303, provides information about the UNICOS kernel configuration files and the run-time configuration files and scripts.
- *UNICOS Networking Facilities Administrator's Guide*, Cray Research publication SG–2304, contains information on administration of networking facilities supported by the UNICOS operating system. This publication contains chapters documenting TCP/IP for the UNICOS operating system, the UNICOS network file system (NFS) feature, and the network information system (NIS) feature.
- *NQE Administration*, Cray Research publication SG–2150, describes how to configure, monitor, and control the Cray Network Queuing Environment (NQE) running on a UNIX system.
- *Kerberos Administrator's Guide*, Cray Research publication SG–2306, contains information on administration of the Kerberos feature, a set of programs and libraries that provide distributed authentication over an open network. This publication contains chapters documenting Kerberos implementation, configuration, and troubleshooting.
- *Tape Subsystem Administration*, Cray Research publication SG–2307, contains information on administration of UNICOS and UNICOS/mk tape subsystems. This publication contains chapters documenting tape subsystem administration commands, tape configuration, administration issues, and tape troubleshooting.

UNICOS/mk system administration publications

This publication is one of a set of related manuals that cover information on the structure and operation of a CRAY T3E computer system running the UNICOS/mk operating system, as well as information on administering various products that run under the UNICOS/mk operating system. This set includes the following publications:

- *UNICOS/mk General Administration*, Cray Research publication SG–2601 contains information on basic administration tasks, file system planning, UNICOS/mk startup and shutdown procedures file system maintenance, basic administration tools, crash and dump analysis, and administration of online features.
- *UNICOS/mk Resource Administration*, Cray Research publication SG–2602 contains information on the administration of various UNICOS/mk features available on CRAY T3E systems. Topics include accounting, global resource

management (GRM), political scheduling, system activity monitoring (SAM), file system quotas, and file system space monitoring.

- *UNICOS/mk Configuration Reference Manual*, Cray Research publication SG–2603 provides details about the administration of UNICOS/mk configuration files created when the UNICOS/mk operating system is installed and configured.
- *UNICOS/mk Networking Facilities Administration*, Cray Research publication SG–2604 contains information on the administration of networking facilities supported by the UNICOS/mk operating system, including TCP/IP for the UNICOS/mk operating system, the UNICOS/mk network file system (NFS) feature, the network information system (NIS) feature, and the Cray-based network monitor.
- *UNICOS/mk Tape Subsystem Administration*, Cray Research publication SG–2607 introduces the Tape Management Facility and the tape interfaces and includes chapters on tape subsystem administration commands, tape configuration, and tape troubleshooting.

Related publications

For tape user information, see the following publication:

- *Tape Subsystem User's Guide*, Cray Research publication SG–2051

The following UNICOS 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 File Formats and Special Files Reference Manual*, Cray Research publication SR–2014
- *UNICOS Administrator Commands Reference Manual*, Cray Research publication SR–2022

- *UNICOS System Libraries Reference Manual*, Cray Research publication SR–2080

The following UNICOS 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

The following manuals are also referenced on man pages in this document:

- *UNICOS Installation Guide*, Cray Research publication SG–2112
- *Application Programmer's Library Reference Manual*, Cray Research publication SR–2165
- *SWS-ION Administration and Operations Guide*, Cray Research publication SG–2204

Design specifications for the UNICOS multilevel security (MLS) feature are based on the trusted computer system evaluation criteria developed by the U. S. Department of Defense (DoD). If you require more information about multilevel security on UNICOS, you may find the following sources helpful:

- DoD Computer Security Center. *A Guide to Understanding Trusted Facility Management* (DoD NCSC-TG-015). Fort George G. Meade, Maryland: 1989.
- DoD Computer Security Center. *Department of Defense Trusted Computer System Evaluation Criteria* (DoD 5200.28-STD). Fort George G. Meade, Maryland: 1985. (Also known as the *Orange book*.)
- DoD Computer Security Center. *Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria* (DoD NCSC-TG-005-STD). Fort George G. Meade, Maryland: 1987. (Also known as the *Red book*.)
- DoD Computer Security Center. *Summary of Changes, Memorandum for the Record* (DoD 5200.28-STD). Fort George G. Meade, Maryland: 1986.
- DoD Computer Security Center. *Password Management Guidelines* (CSC-STD-002-85). Fort George G. Meade, Maryland: 1985.

- Wood, Patrick H. and Stephen G. Kochan. *UNIX System Security*. Hasbrouck Heights, N.J.: Hayden Book Company, 1985.

  Note: If your site wants to purchase the optional SecurID card used with UNICOS MLS network security, the necessary hardware, software, and user publications can be obtained from Security Dynamics, Inc., 2067 Massachusetts Avenue, Cambridge, MA, 02140, (617) 547-7820.

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.

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.

Tape Subsystem Administration [1]

This chapter introduces the tape interfaces and administration commands.

1.1 Tape interfaces

The tape subsystem supports two interfaces: the tape daemon-assisted interface and the character-special tape interface. This manual describes the tape daemon-assisted interface, which is referred to as the tape subsystem throughout the manual. It is also called the Tape Management Facility.

The character-special tape interface and the tape daemon-assisted interface may operate concurrently. Devices for both interfaces are defined in the same configuration file and are defined identically; that is, the interface is not identified in the configuration file. For information on the character-special tape interface, see the *Tape Subsystem User's Guide*, Cray Research publication SG–2051.

Warning: Starting with the UNICOS 10.0 release, the term *Cray ML-Safe* replaces the term *Trusted UNICOS*, which referred to the system configuration used to achieve the UNICOS 8.0.2 release evaluation. Because of changes to available software, hardware, and system configurations since the UNICOS 8.0.2 system release, the term *Cray ML-Safe* does not imply an evaluated product, but refers to the currently available system configuration that closely resembles that of the evaluated Trusted UNICOS 8.0.2 system.

For the UNICOS 10.0 release, the functionality of the Trusted UNICOS system has been retained, but the `CONFIG_TRUSTED` option, which enforces conformance to the strict B1 configuration, is no longer available.

1.2 Administration commands

This section briefly describes tape subsystem administration commands common to all Cray Research systems.

`tpapm`(8) — The `tpapm`(8) command requests that the tape daemon mount a volume (identified by the volume serial number (VSN)) on any available drive that may be serviced by an autoloader. This function is useful when you must premount volumes required by some program (for example,

	a backup) that is performing a multivolume operation, and the required volumes reside in the autoloader storage unit.
`tpbmx`(8)	The `tpbmx`(8) command displays tape device information that may be useful to operators and system administrators. This information comes primarily from the `tpdtab` structure for each device in the kernel (the definition of `tpdtab` is in the file `/usr/include/sys/tpd.h`).
`tpclr`(8)	The `tpclr`(8) command clears the last pending request on the data path to a tape drive. All tables and data associated with that device are cleared, if possible.
`tpconf`(8)	The `tpconf`(8) command converts a tape daemon configuration file to binary format for the tape daemon to process when it is started.
`tpconfig`(8)	The `tpconfig`(8) command configures tape devices up and down, changes the status of the associated media loaders, and assigns a media loader to a device and reassigns a device group to a device.
`tpcore`(8)	The `tpcore`(8) command initiates the tape subsystem monitor.
`tpdaemon`(8)	The `tpdaemon`(8) command starts the tape daemon. It provides the routing and control of the various components used in tape resource management, device management, volume mounts and dismounts through operator communication or autoloader requests, label processing, volume switching, accounting, security, and error recovery.
`tpdev`(8)	The `tpdev`(8) command displays the status of tape devices and associated components of the tape data path.
`tpdstop`(8)	The `tpdstop`(8) command stops the tape daemon in an orderly fashion.
`tpformat`(8)	The UNICOS `tpformat`(8) command reserves the specified resource, mounts the requested

	volume, and issues the volume format request to the ER90 device.
`tpfrls`(8)	The `tpfrls`(8) command lets the operator release the tape reservations made by a user. `tpfrls`(8) also clears all active tape devices and kills the user processes using the tape devices.
`tpgstat`(8)	The `tpgstat`(8) command displays the reservation status for each device group in the system for each tape user.
`tpinit`(8)	The `tpinit`(8) command initializes the tape subsystem.
`tplabel`(8)	The `tplabel`(8) command labels tapes and may perform other functions depending on the interface being used.
`tpmls`(8)	The `tpmls`(8) command displays the status of the tape loaders in the system.
`tpmql`(8)	The `tpmql`(8) command displays the current mount request list for all users who have completed initial mount processing and have a mount request pending.
`tpscr`(8)	The `tpscr`(8) command returns volumes allocated by a user to the loader scratch pool.
`tpset`(8)	The `tpset`(8) command sets features for the tape daemon. It changes the status of automatic volume recognition (AVR), the status of front-end servicing (FES), the status of the Cray/REELlibrarian (CRL), the status of overcommitted mount requests, the status of tracing for the tape daemon, or the destination of tape operator messages issued by the tape daemon. All of these features except tracing can be set in the tape configuration file. `tpset`(8) returns the current status of all features.
`tpu`(8)	The `tpu`(8) command is used by the system operator to unload tapes. This command has no effect on a tape that is currently in use. It is most

useful for unloading tapes and freeing tape drives on systems running with automatic volume recognition (AVR).

Tape Configuration [2]

This chapter provides information on UNICOS and UNICOS/mk configuration.

2.1 UNICOS configuration

In order for users to access tapes through the tape daemon-assisted interface or through the character-special tape interface, the `roptions` file must contain a tape entry. The following entry is required to activate the tape software:

```
RC-TAPE="YES"
```

Tape configuration is usually maintained through the UNICOS installation and configuration menu system, which uses the `text_tapeconfig` file. You should set the parameters in the file to values that suit your system. This section contains information on configuration settings and the system boot parameter file.

2.1.1 Configuration settings

Tape configuration on a UNICOS system is determined at the following four levels. The higher levels override the lower levels if the settings differ.

1. Kernel generation. Kernel parameters that relate to tapes are located in `config.h`, a Cray Research systems source file. It provides defaults for the `TAPE_MAX_CONF_UP` and `TAPE_MAX_PER_DEV` parameters in the system boot parameter file.

2. System boot. The system boot parameter file specifies tape parameters in `~cri/os/uts/param`, an operator work station (OWS) file, in the `/sys/param` file for the CRAY J90 series systems with IOS-V, or in `/opt/CYRIOS/`*systemname*`/param`, a system workstation (SWS) file.

3. Tape system initialization. The `tpinit`(8) command reads and processes the parameters you have specified in `/etc/config/text_tapeconfig`. For parameter descriptions, see the `text_tapeconfig`(5) man page. This page also contains tape configuration file examples for GigaRing based systems and for systems with I/O subsystems Model E (IOS-E) and Model V (IOS-V).

4. Administration commands. You may specify particular settings and options by using tape administration commands such as `tpset`(8) to make run-time changes. For more information, see Chapter 1, page 1.

Tape configuration parameters that you can modify are defined in the system boot parameter file and in the tape configuration file on Cray Research systems.

Note: For Trusted UNICOS systems, specific configuration values must be observed. See "Configure the UNICOS 9.0 System" section in the *UNICOS Installation Guide*, Cray Research publication SG–2112, for more information.

2.1.2 System boot parameter file

The system boot parameter file specifies parameters that define the kernel. The following three parameters are tape-related:

`TAPE_MAX_CONF_UP` This parameter defines the maximum number of tape devices that can be configured up. The tape configuration file may specify more devices than are specified in this parameter. The default value is 4.

`TAPE_MAX_PER_DEV` This parameter defines the maximum number of bytes that a device can have for buffers. This number is rounded up to the next multiple of 4096 bytes by the tape driver. The default value is 65536 bytes.

`TAPE_MAX_DEV` This parameter specifies the maximum number of tape devices in the system. The following define the types of tape devices:

Real devices	Tape devices upon which you can mount a tape.
Pseudo devices	These devices are not real devices. They are used by the tape daemon and tape users as handles to gain access to the tape driver. A pseudo device may be opened and the

resulting file descriptor used to issue `ioctl`(2) system calls to the tape driver.

The value specified by this parameter is the sum of all of the following:

- The number of real tape devices
- The number of tape I/O processors (IOPs)
- The total number of channels for tapes

The default value is 32.

The following example shows the section of the system boot parameter file that specifies a maximum of 40 tape devices for the IOS-E:

```
unicos {
        3536 NBUF;
        0 LDCHCORE;
        32  TAPE_MAX_CONF_UP;     /* Maximum of 32 tapes configured up */
        196608 TAPE_MAX_PER_DEV;  /* Maximum size of buffer per device */
                                  /*  (in bytes) */
        40 TAPE_MAX_DEV           /* Maximum of 40 tape devices */
        500 NLDCH;
        24 PDDMAX;
        256 LDDMAX;
        256 PDDSLMAX;
        256 MDDSLMAX;
        256 SSDDSLMAX;
        256 SDDSLMAX;
}
```

2.2 UNICOS/mk configuration

Tape configuration is usually maintained through the UNICOS/mk configuration tool, which uses the `text_tapeconfig` file. You should set the parameters in the file to values that suit your system. This section contains information on configuration settings and the parameter file.

2.2.1 Configuration settings

The following settings affect tape usage on a UNICOS/mk system:

1. Kernel generation. The system administrator should make sure that the tape flag in the CRAY T3E `/etc/config/config.mh` file is set to 1. This flag controls whether or not a site will have the online tape capability in the kernel:

```
/*
 *  If set to one (1), the CONFIG_TAPE variable turns on the online tape
 *  capability in the kernel. If set to zero (0), the code is turned
 *  off. Its default is one (1), or ON.
 */

#ifndef CONFIG_TAPE
#define CONFIG_TAPE 1
#endif
```

2. `rcoptions` file set-up. The system administrator should also make sure that the CRAY T3E `rcoptions` file contains the following tape entry:

   ```
   RC_TAPE="YES"
   ```

   This entry is required to activate tape software so that users can access tapes. It is needed for either the tape daemon-assisted interface or the character-special tape interface.

   For information on startup and shutdown, see *UNICOS/mk Configuration Reference Manual*, Cray Research publication SG–2603.

3. System startup. The system administrator can specify tape parameters in the SWS `/opt/CYRIos/<mainframe_name>/param` file.

4. Multiuser initialization. The `tpinit`(8) command reads and processes the parameters specified in the `/etc/config/text_tapeconfig` file. For a description of these parameters, see the `text_tapeconfig`(5) man page.

   The tape daemon uses these values to decide what to do in various situations. The tape configuration file must be updated to configure the tape daemon before the daemon is started.

   The tape configuration file defines all tape hardware used by the system. The diagnostic devices are implicitly defined when the nodes and the channels are defined. You must not define them again.

5. Administration commands. Once the system is in multiuser mode, the system administrator may specify particular settings and options by using tape administration commands such as `tpset`(8) to make run-time changes. For a brief description of the commands, see Chapter 1, page 1.

2.2.2 Parameter file

The parameters in the parameter file define the kernel. The following three parameters are tape-related:

`tpd_max_bufz` — Specifies the maximum tape buffer size in bytes for each tape device configured up. Accepts 0 to 999999999. This value is rounded up to the next multiple of 4096 bytes.

`tpd_max_conf_up` — Specifies the maximum number of tape devices that may be configured up. Accepts 0 to 9999.

`tpd_maxdev` — Specifies the maximum number of tape devices that may be configured in the tape subsystem. Accepts 0 to 9999. The following define the types of tape devices:

Real devices	Tape devices upon which you can mount a tape.
Pseudo devices	These devices are not real devices. They are used by the tape daemon and tape users as handles to gain access to the tape driver. A pseudo device may be opened and the resulting file descriptor used to issue `ioctl`(2) system calls to the tape driver.

The value specified by this parameter is the sum of all of the following:

- The number of real tape devices
- The number of tape nodes
- The total number of channels for tapes

The default value is 32.

For more information, see *SWS-ION Administration and Operations Guide*, Cray Research publication SG–2204.

The following example shows the section of the parameter file that specifies a maximum of 32 tape devices.

```
tapedriver.tpd_maxdev           = 32;
tapedriver.tpd_max_conf_up      = 32;
tapedriver.tpd_max_bufz         = 262144;
```

Other Tape Administration Issues [3]

This chapter describes the following important issues related to the tape subsystem administration:

- Naming and numbering device groups
- User database (UDB) considerations
- User exits
- Tape autoloaders
- Message daemon and operator interface

3.1 Naming and numbering device groups

Device groups are communicated to all relevant subsystems; use care in naming and numbering the device groups. The subsystems (such as the user database (UDB), the Network Queuing System (NQS), the accounting system, `dump`(2) and `restore`(8), and data migration) that use the tape subsystem have different internal definitions for tape device groups. Any change (from `CART` and `TAPE`) to the names of the device groups will probably affect one of these subsystems. Therefore, you should refer to the appropriate subsystem documentation before changing or adding device group names.

The order of the device groups can be defined in the `/etc/config/text_tapeconfig` file with the `DEVICE_GROUP` statement. The order is determined from the devices defined by using the `DEVICE` statement. Job limit checking is based on the order of device groups in the tape configuration file.

3.2 User database considerations

The user database (UDB) contains several fields that are important to the user's ability to access tapes. The `permbits` field is directly accessed by the tape subsystem during the execution of a `tpmnt`(1) command. In order for a user to mount a tape with a label type of `blp` (bypass label processing), the user must have the `bypasslabel` permission bit set. Any application that intends to use high-speed positioning which bypasses tape daemon control, must also have the `tape-manage` permission bit set.

The tape administrator establishes batch and interactive tape limits by setting the appropriate entries in the `jtapelim` table. The tape daemon restricts accesses to device groups based on the values passed during job initiation. The entries of `jtapelim` table correlate one for one with the device group displayed with the `tprst`(1) command. The order of the device groups in the tape configuration file determines the order in `tprst`(1) output.

The following example shows limits for batch tapes listed in a UDB:

```
jtapelim[b][0]  :02:
jtapelim[b][1]  :01:
jtapelim[b][2]  :00:
jtapelim[b][3]  :00:
jtapelim[b][4]  :00:
jtapelim[b][5]  :00:
jtapelim[b][6]  :00:
jtapelim[b][7]  :00:
```

If the tape configuration file defined device groups of `CART`, `TAPE`, `3490`, and `TEST`, then a corresponding `tprst`(1) command would show the following:

```
dev grp  w     rsvd     used available
CART             0        0    0
TAPE             0        0    0
3490             0        0    0
TEST             0        0    0
```

In this example, the user can submit a batch job that is limited to accessing 2 `CART` devices and 1 `TAPE` device.

3.3 User exits

User exits allow users to add special routines to communicate with the tape daemon without having access to the source. User exits allow a system process to examine and modify a structure associated with a tape file.

3.3.1 Implementation

To implement an user exit, it is necessary to modify and recompile the `tpuex.c` file and to switch the user exit on or off within the `/etc/config/text_tapeconfig` file. To switch the individual user exit or all user exits (`UEX_ALL`) on or off, make an entry in the tape configuration file.

If you are using the Configuration Tool (CT), select `Tapes` from the CT's subsystem list and either load an existing configuration file or create a new configuration file. User exit options are selected from the following CT menu:

```
Tape Configuration
.  Select Tape SubSystem Options
   .   General Options
```

The following is an example of the entry to add to the `OPTIONS` statement of the tape configuration file:

```
user_exit_mask = (UEX_ASK_EXPDT,UEX_ASK_LBSW,UEX_ASK_RETRY),
```

The user exits for `user_exit_mask` are as follows:

User exit	Description
`UEX_ALL`	Enables all user exits
`UEX_ASK_EXPDT`	Enables `uex_askexpdt` user exit
`UEX_ASK_HDR1`	Enables `uex_ask_hdr1` user exit
`UEX_ASK_LBSW`	Enables `uex_asklbsw` user exit
`UEX_ASK_RETRY`	Enables `uex_askretry` user exit
`UEX_ASK_VERSCR`	Enables `uex_askverscr` user exit
`UEX_ASK_VSN`	Enables `uex_askvsn` user exit
`UEX_ASK_SCR_VSN`	Enables `uex_scr_vsn` user exit
`UEX_CHK_ACCESS`	Enables `uex_chk_access` user exit
`UEX_CLS_FILE`	Enables `uex_cls_file` user exit
`UEX_MAC_HDR2`	Enables `uex_mac_hdr2` user exit
`UEX_MNT_MSG`	Enables `uex_mnt_msg` user exit
`UEX_SM_DEX`	Enables `uex_sm_dex` user exit
`UEX_SM_DUX`	Enables `uex_sm_dux` user exit
`UEX_SM_VAX`	Enables `uex_sm_vax` user exit
`UEX_SM_VUX`	Enables `uex_sm_vux` user exit
`UEX_START`	Enables `uex_start` user exit

`UEX_STOP` Enables `uex_stop` user exit

If an invalid option is used, an error message appears in the `daemon.stderr` file similar to the following:

```
TM425 - file ./text_tapeconfig, line 311 at ",", offset 54 : syntax error:
expecting: KEYWORD_PARAM-VALUE ....
```

To find the error, check the appropriate line in the `text_tapeconfig` file. The `at ","` in the error message indicates that the tape daemon does not expect to see the character `","` here.

There are two other files, `tpuex.c` and `tpuex.h`, that are necessary to implement user exits. They are located in the directory `/usr/src/cmd/c1/tp/tpnex`. The file `tpuex.c` has stub routines corresponding to each user exit; the `tpuex.h` file contains the declarations needed for defining the user structure (`uex_table`).

All user exits have access to the `uex_table` structure. An exit may make its decisions based on the values contained in this structure.

To use the user exits, follow these three steps:

1. Modify the `tpuex.c` file to reflect the required action of the user exit.
2. Recompile the `tpnex.c` file using the `nmake` command in the directory `/usr/src/cmd/c1/tp`. This creates a new `tpnex.o` file, relinks the file, and creates new executables.
3. Install the new tape daemon with the `nmake install` command.

Examples on how to code the user exits can be found in the `tpuex.c` file.

3.3.2 User exit descriptions

User exits returning the values of 0 or -1 can use the defined symbolic values of `YES` (0) or `NO` (-1) defined in the `tpuex.h` file. Descriptions of tape subsystem user exits follow:

`uex_askexpdt(uex_table)`

Receives the `uex_table` structure and returns an integer value of `YES` or `NO`.

This exit returns an answer to the question "Can user *userid* write on unexpired VSN *vsn*?"

`uex_asklbsw(uex_table)`

Receives the `uex_table` structure and returns an integer value of `YES` or `NO`.

This exit returns an answer to the question "Can user *userid* switch from label *original label* to *new label* for VSN *vsn*?"

`uex_askretry(uex_table, message_type, server_or_front-end, message_id, reason_for_retrying)`

Receives the `uex_table` structure and the additional parameters as shown above and returns an integer value of `YES` or `NO`.

It is called when the tape daemon is unable to send a request to a front-end/server and returns an answer to the question "Should message to front end be re-sent or aborted?"

The returned value of `YES` means to retry the request; `NO` means to cancel the request.

`uex_askverscr(uex_table, vsn)`

Receives the `uex_table` structure and a VSN. It returns an integer value of `YES` or `NO`.

This exit returns an answer to the question "Is volume *vsn* on device *dvn* a valid scratch volume for the job ID *jid*?"

`uex_askvsn(uex_table)`

Receives the `uex_table` structure and returns either a character pointer with the value `NULL` or an address of a string.

This exit returns an answer to the question "What is the VSN on device *dvn*?"

The returned value of `NULL` means no VSN was returned while a pointer to a string is used as the value of the scratch VSN. If no VSN is returned, then the tape daemon calls the `askvsn`() routine, just as if the user exit had not been taken.

`uex_ask_hdr1(uex_table)`

Receives the `uex_table` structure and returns an integer value of `YES` or `NO`.

This user exit is called from a child process in the tape daemon at the point where the VOL1 and HDR1 labels have been read from a tape and the child process prepares to check some of the values in the HDR1 label against values that are kept by the tape daemon and its child processes.

A site can use this user exit to add code that enables the tape daemon to do the following:

- Obtain a number that controls how many characters of the file identifier field in a HDR1 label is compared to a character string kept by the tape daemon or, to an alternate character string that is provided by this user exit.

  The tape daemon uses the number in the `user_fidl` field of the `uex_table` structure. If the site changes this number, the modified number must have a value that is equal to or greater than 1 and less than or equal to 16. The number must be returned in the `user_fidl` field, while the return value from this user exit must be `YES`. If `NO` is returned, the `user_fidl` field is not examined.

- Obtain an alternate character string for the file identifier to be compared to the character string in the file identifier field in the HDR1 label from the tape.

  The character string that the tape daemon uses is in the `user_fid` field of the `uex_table` structure. If the site changes this string, the modified character string must be stored in the `user_fid` field, while the return value from this user exit must be `YES`. If `NO` is returned, the `user_fid` field is not examined.

- Obtain an alternate one character string, which, in case of ANSI labels, is compared to the accessibility character string in the accessibility field in the HDR1 label from the tape. If the character strings match, the action that is taken is the same as the action taken for the space character as defined in the ANSI standard.

The character string that the tape daemon uses is in the `user_vac` field of the `uex_table` structure. If the site changes this string, the modified character string must be stored in the `user_vac` field, while the return value from this user exit must be `YES`. If `NO` is returned, the `user_vac` field is not examined.

If this user exit returns `YES`, the following three actions occur:

- The `surfeited` field is checked for a number that is equal to or greater than 1 and less than or equal to 16. If the returned number is outside this range, the default value of 17 is used.
- The contents of the `surfed` field is copied into a tape daemon structure.
- The contents of the `served` field is copied into a tape daemon structure after it is checked against the following characters:

  ```
  A ... Z, 0 ... 9, " !\"%&'()*+,-./:;<=>?_"
  ```

If `NO` is returned, `uex_table` information is not used to update data structures in the tape daemon.

`uex_chk_access(uex_table)`

Receives the `uex_table` structure and returns an integer value of `YES` or `NO`.

This user exit is called from a child process in the tape daemon at the point where the tape daemon has accepted a tape volume to read from or to write to. It enables a site to add code to check the access of the tape.

For example, the code could allow or reject access to a tape volume after it has checked a locally maintained permission file. When this user exit checks permission to access an output tape, the tape daemon upon return from this user exit checks the `user_error` field of the `uex_table` structure. If this field contains error code `ETNSC` (90089 - not scratch), the tape daemon rejects the tape volume. If more tape volumes have been specified in the `tpmnt`(1) command, `tpmnt`(1) tries the next tape volume.

Besides setting the user_error field to ETNSC, this user exit also sets the uex_table bit field uex_lst.flg.nsc to 1. The tape daemon updates the fit field lst.flg.nsc with this information from the uex_table field.

If the user_error field contains any other error number, the tape daemon upon return aborts the child process with error code EACCES (13 - permission denied). If this user exit returns the value NO, the tape daemon aborts the child process with error code EACCES. If this user exit returns the value YES, the tape daemon accepts the tape volume and processing continues.

When this user exit checks permission to access an input tape, the tape daemon upon return from this user exit checks the return code. If the return code is NO, the tape volume is rejected and the child process aborts with error code EACCES. If the return code is YES, the tape volume is accepted and processing continues.

Besides the possible update of bit field lst.flg.nsc in the fit, no other information from the uex_table is used to update data structures in the tape daemon.

uex_cls_file(uex_table)

Receives the uex_table structure and returns. The tape daemon upon return from this user exit does not update any of its information with information from uex_table.

This user exit is called from a child process in the tape daemon after the tape processing is completed and when the tape file is about to be closed. The exit provides a site with an opportunity to add code. For example, the code could enable the tape daemon to add information to the tape.msg file concerning the tape volumes that were used while processing the tape file.

uex_mac_hdr2(uex_table)

Receives the uex_table structure and returns an integer value of YES or NO.

This user exit is called from a child process in the tape daemon after the tape daemon reads the label information from the tape and has called the security code to check proper beginning-of-tape structure: VOL1, HDR1 and HDR2 labels. The user exit is called when a tape has a VOL1 and a HDR1

label, but not a HDR2 label. A site may use this user exit to add code that allows or rejects access to the tape volume.

If this user exit returns the value `NO`, the tape daemon continues its normal processing. It allows the tape to be overwritten, but not to be read. If the exit returns the value `YES`, the tape daemon allows the user access to the tape. A return code of `NO` complies with security guidelines.

No information from the `uex_table` structure is used to update data structures in the tape daemon.

`uex_mnt_msg(uex_table)`

Receives the `uex_table` structure and returns.

This user exit is called from either a child process or the tape daemon itself after the tape daemon has built a tape mount message and before it is sent to be processed. A site can use this exit to add code that supplements information in the existing mount message or changes it in some other way. When the tape daemon transfers control to this user exit, the `user_buff` field of the `uex_table` structure contains the address of the mount message character string and the `user_bytes` field of the `uex_table` structure contains the length in bytes of the memory block that are allocated to hold the mount message.

If this user exit extends the length of the delivered character string beyond the size of the allocated memory block, the user exit allocates the necessary memory to store the newly composed mount message. The address of which is returned to the tape daemon in the `user_buff` location. The length in bytes of the newly allocated memory block is returned in the `user_bytes` field. The `uex_table` field update is set to a nonzero value.

If this user exit does not extend the length of the delivered character string beyond the size of the allocated memory block, the code does not allocate another memory block, and the address in the `user_buff` field is left unaltered. The length in bytes of the allocated memory block in the `user_bytes` field also remains unaltered. The `update` field of the `uex_table` structure is set to zero.

When this user exit returns, the tape daemon checks the value of the `update` field. If its value is zero, the tape daemon continues its normal processing. It sends the mount message from the location it has allocated to be processed. If the value in the `update` field is nonzero, the tape daemon compares the address of the memory block that it has allocated for its mount message and the address that has been returned in the `user_buff` field. If these addresses are the same, the tape daemon continues its normal processing. If these addresses differ, the tape daemon frees the memory block it had allocated for its mount message and takes the address from the `user_buff` field as its replacement.

No other `uex_table` information is used to update data structures in the tape daemon.

`uex_scr_vsn(uex_table)`

Receives the `uex_table` structure and returns either a character pointer with the value `NULL` or an address of a string.

This exit allows a site to specify the VSN for a scratch request.

The returned value of `NULL` means no VSN was returned while a pointer to a string is used as the value of the scratch VSN. If no VSN is returned, the tape daemon uses the default scratch VSN, just as if the user exit had not been taken.

`uex_sm_dex_1(uex_table)`

Receives the `uex_table` structure and returns.

This user exit is called from a child process in the tape daemon after the tape daemon has built a dataset enquiry (dex) request for a servicing front-end machine and before it is sent to be processed. A site can use this user exit to add code to supplement information in the existing dex request or change it in some other way.

When the tape daemon transfers control to this user exit, the `user_buff` field of the `uex_table` structure contains the address of the dex request and the `user_bytes` field of the `uex_table` structure contains the length in bytes of the memory block that is allocated to hold the dex request. The `/usr/src/cmd/c1/tp/tpuex/festbls.h` header file

contains a layout of the data structures making up the format for the delivered dex request.

If this user exit extends the length of the delivered dex request beyond the size of the allocated memory block, the code allocates the necessary memory to store the newly composed dex request. The address of this allocated memory block is returned to the tape daemon in the `user_buff` location. The length in bytes of the newly allocated memory block is returned in the `user_bytes` field. The length in words of the newly composed dex request is returned in the `user_wc` field of the `uex_table` structure. The `update` field of the `uex_table` structure must be returned set to a nonzero value, while the `yes_no` field must be returned set to `YES`.

If this user exit does not extend the length of the delivered dex request beyond the size of the allocated memory block, the code does not have to allocate another memory block and the address in `user_buff` field remains unaltered. The length in bytes of the allocated memory block in the `user_bytes` field also remains unaltered. The length in words of the newly composed dex request is returned in the `user_wc` field. The `update` field must be returned set to a nonzero value, while the `yes_no` field must be returned set to `YES`. If this user exit does not change the delivered dex request in any way, the `update` field must be returned set to zero, while the `yes_no` field is returned set to `YES`. If this user exit determines that the dex request should not be sent to the servicing front-end machine, the `yes_no` field is has to be returned set to `NO`.

When this user exit returns, the tape daemon checks the value returned in the `yes_no` field. If the value in this field is `NO`, the tape daemon does not send the request to the servicing front-end machine and continues its processing.

If the value returned in the `yes_no` field is `YES`, the tape daemon checks the value of the `update` field. If its value is zero, the tape daemon continues its normal processing. It sends the dex request from the location it has allocated to the servicing front-end machine to be processed. If the value in the `update` field is nonzero, the tape daemon replaces its value of the length in words of the dex request with the value which is returned in the `user_wc` field. It compares the address of the memory block it has allocated for its dex request and the

address which has been returned in the `user_buff` field. If these addresses are the same, the tape daemon continues its normal processing. If these addresses differ, the tape daemon frees the memory block it allocated for its dex request and takes the address from the `user_buff` field as its replacement, after which it continues its normal processing.

No other `uex_table` information is used to update data structures in the tape daemon.

`uex_sm_dux_2(uex_table)`

Receives the `uex_table` structure and returns.

This user exit is called from a child process in the tape daemon at the point where the tape daemon receives a reply from the servicing front-end machine to a dataset enquiry (dex) request and before it processes this reply. A site can use this exit to add code that processes the reply in accordance with local requirements. When the tape daemon transfers control to this user exit, the `user_buff` field of the `uex_table` structure contains the address of the dex reply and the `user_bytes` field contains the length in bytes of the memory block which has been allocated to hold the dex reply. The `/usr/src/cmd/c1/tp/tpuex/festbls.h` header file contains a layout of the data structures making up the format of the delivered dex reply.

If this user exit determines that the servicing front-end machine has returned a message in the reply, the address of the message must be returned to the tape daemon in the `user_tmsgp` field of the `uex_table` structure. It assures the message is properly processed. If the servicing front-end machine has not returned a message in the reply field, `user_tmsgp` has to be zero.

The `user_error` field is provided in case the user exit encounters an error condition which has to abort the tape daemon child process. When the user exit returns this field is set to a nonzero value and the `yes_no` field of the `uex_table` structure set to value `NO`, the tape daemon passes the value in `user_error` on to the abort function. For a list of possible return values, see the *Tape Subsystem User's Guide*, Cray Research publication SG–2051.

If this user exit completes without errors and updates information in the `uex_table` structure from the information delivered in the dex reply, it returns a nonzero value to the tape daemon in the `update` field of the `uex_table` structure and in the `yes_no` field of the `uex_table` structure value `YES`. This causes various data structures in the tape daemon to be updated with `uex_table` information. The code in function `uex_sm_dex_2()` in the `/usr/src/cmd/c1/tp/tpuex/tpuex.c` file contains an example which is based on the way the tape daemon processes the dex reply. It shows the `uex_table` fields which have to be updated. If this user exit completes without updating the `uex_table` fields and has determined that the tape daemon has to do the processing of the dex reply, it returns to the tape daemon with the `yes_no` field set to `YES` and the `update` field set to zero. This prevents the tape daemon from updating its data structures with information from the `uex_table` structure. If the user exit relies on the tape daemon to process the dex reply, the reply must be in the format the tape daemon can handle.

`uex_sm_vax(uex_table)`

Receives the `uex_table` structure and returns an integer value of `YES` or `NO`.

This exit returns an answer to the question "Can user *uid* access volume *vsn*?" It is called in place of the volume access request made to the front-end system.

This routine must validate access for a VSN and set the following:

- Expiration information for the file
- The allowed-permission-bits structure

Returning a nonzero value denies access to the dataset.

`uex_sm_vux(uex_table)`

Receives the `uex_table` structure and returns an integer value of `YES` or `NO`.

This exit provides the opportunity to update tables and log fields after a volume has been accessed. It is called in place of the volume access request made to the front-end system.

The return value `YES` means that the update was successful, while the return value `NO` means that the update failed.

In addition, the new user exit, `uex_vsn_ok_to_use`, is called prior to sending a tape mount request to the tape daemon. It provides the requested VSNs for the tape mount. A site may replace the default exit with an exit tailored for its needs.

3.4 Automatic volume recognition

A tape daemon with automatic volume recognition (AVR) enabled automatically reads the label of a tape mounted on a device that is configured up but is not assigned to a user. The tape daemon saves the volume serial number (VSN); consequently, when the tape is requested by a `tpmnt`(1) command, the correct tape is assigned. When a nonlabeled tape is mounted, the operator is asked to supply the VSN.

For ER90 devices, the tape daemon saves the format ID. The correct tape is assigned based on this format ID. When a blank tape is mounted, the operator is asked to supply the external ID.

You can dynamically turn the AVR option on and off by issuing the `tpset`(8) command. To turn AVR on, enter the following command:

```
tpset -a on
```

To turn AVR off, enter the following command:

```
tpset -a off
```

When the tape daemon successfully switches the AVR option, no message is issued. If the tape daemon is busy with active tape users, you will receive a message, and the AVR option will not be switched.

Issuing `tpset`(8) with no options results in a report of the status of AVR front-end servicing, the status of front-end servicing, the status of tape tracing, the status of the volume management facility, the status of Cray/REELlibrarian, and tape operator ID.

When a user asks for a tape, the tape daemon verifies that the requested tape is mounted on a drive in the requested device group and that it is not already

assigned to a user. If the tape is found, it will be assigned to the user who issued the request. If the tape is not found, a message will be sent to the operators requesting that the tape be mounted.

When the tape daemon is looking for a device to assign to a user, it stops the search as soon as a matching VSN for BMX devices or format ID for ER90 devices is found. If multiple tapes with the same VSN are mounted, the first drive that has a matching VSN and is not already assigned is assigned to the user. Others with a matching VSN are queued. Therefore, it is best to have a unique VSN for all tapes, including scratch tapes.

AVR allows the operator to mount a tape requested by a user on one of the following devices:

- Any drive that is not assigned to a tape user of the requested device group. The output of the `tpstat`(1) command shows the drives that are unassigned.
- The drive assigned to the user. The operator can determine the drive assigned to a given user by looking at the output of the `tpstat`(1) command. The drive assigned to the user is indicated by the VSN and an asterisk (*) preceding the VSN.

When a device is released (and neither the `-n` option of the `rls`(1) command nor the `-u` option of the `tpmnt`(1) command has been specified), the tape is unloaded.

If the user asks for a scratch tape (by failing to specify a VSN on the `tpmnt`(1) command), the tape daemon asks for a mount with the default scratch VSN. The operator must specify the name of a device that has a scratch tape mounted in the reply to the mount message. The tape daemon then assigns the drive to the user.

After a tape drive is configured up and a tape is mounted, the operator should not unload the tape by pressing the "reset" button or the "not ready" button. A tape should be unloaded by use of the `tpu`(8) command. The `tpu`(8) command unloads tapes only on drives that are configured up but not assigned.

3.5 Tape autoloaders

This subsection briefly describes the command flow between the software components that are involved in the handling of tape volumes through an autoloader. This flow is described in general and then in detail for StorageTek, EMASS, and IBM autoloaders. This subsection also describes how to install and

configure each type of autoloader; and describes some pointers in organizing, numbering, naming and accessing tape devices.

Note: EMASS autoloaders are only supported on UNICOS systems.

3.5.1 Command flow

The following steps describe command flow between software components that are involved in the handling of tape volumes.

1. The tape daemon starts a child process for each autoloader that has been configured up. This child process is a daemon process that executes as long as the autoloader it is associated with is configured up. If this autoloader is configured down, the child process is terminated.
2. The tape daemon sends requests to the child process over a named pipe; the tape daemon receives a confirmation reply and a final reply for each of these requests. The final reply contains status information concerning the completion of the request.
3. After the child process has received a request from the tape daemon and has returned a confirmation reply, it translates the request into a format that is acceptable to software supplied by the autoloader vendor. This software is installed on a computer platform that is connected to a network to which the Cray Research system is connected.
4. The translated request is sent to the vendor-supplied software using the Remote Procedure Call/eXternal Data Representation (RPC/XDR) protocol by way of Transmission Control Protocol/Internet Protocol (TCP/IP).
5. The vendor software returns a confirmation reply after the request has been received from the Cray Research system.
6. After the vendor-supplied software has completed the request, it sends a final reply to the Cray Research system over the network. The reply contains status information concerning the completion of the request.
7. When the child process receives the reply from the vendor-supplied software, it translates this reply into a format that is acceptable to the tape daemon and sends it over a pipe to the tape daemon.

This process is depicted in Figure 1.

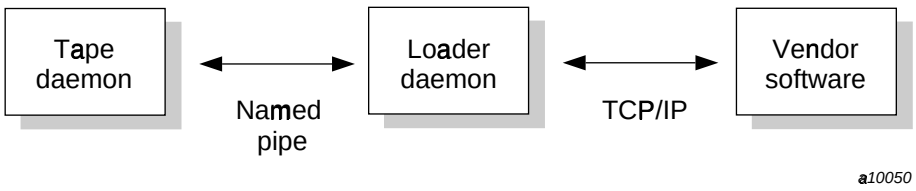

Figure 1. UNIX autoloader communication

3.5.2 StorageTek autoloader information

The StorageTek Autoloader Cartridge Systems (ACS) support the IBM 3480, 3490, and 3490E compatible tape cartridges as well as the cartridges used for the helical-scan StorageTek Redwood drives. These systems use the StorageTek 4480 drive, StorageTek Silverton drive, StorageTek Timberline drive, and StorageTek Redwood drive.

The child process started by the tape daemon for a StorageTek autoloader is named `stknet`, and the vendor-supplied software is called Automated Cartridge System Library Server (ACSLS). The ACSLS application typically runs on a Sun platform.

`stknet` is a program in executable binary format for which the building blocks are delivered with the release materials. The `cmd/c1/tp/stkacs` directory contains these building blocks:

File	Description
`Nmakefile`	Contains instructions to create `stknet` out of `stknet.o` and `stklib.a`. It is installed in the `cmd/c1/tp/stkacs` directory. This `Nmakefile` file is, typically, called into execution from the `Nmakefile` located in the `cmd/c1/tp` directory. When this `Nmakefile` file is called into execution with the `install` parameter, `stknet` is placed in the `/usr/lib/tp` directory from which the tape daemon calls it into execution.
`stklib.a`	Contains an archive of modules in relocatable binary format that link into the executable version of `stknet`.

`stknet.o`	Contains a module in relocatable binary format.

`stknet` supports all tape drives that can be connected to a StorageTek library. If `stknet` serves StorageTek Redwood drives, license CRSTK/STKRED and its accompanying Cray FLEXlm key is required to activate the code in `stknet` that supports the StorageTek Redwood drives. For assistance with the license, contact your Cray Research service representative. No license is needed for all other tape drives that can be connected to a StorageTek library.

3.5.3 IBM autoloader information

The IBM 3494 Tape Library Dataserver supports the IBM 3480, 3490, and 3490E tape cartridges and uses the drives that IBM supports with these autoloaders.

The child process started by the tape daemon for an IBM autoloader is called `ibmnet` and the vendor-supplied software is called Controlled Path Service (CPS). The `atl` application typically runs on an IBM RISC system/6000 platform.

`ibmnet` is a program in executable binary format for which the building blocks are delivered with the release materials. The directory `cmd/c1/tp/ibmtld` contains these building blocks:

File	Description
`Nmakefile`	Contains instructions to create `ibmnet` out of `ibmnet.o` and `ibmlib.a`. It is installed in the `cmd/c1/tp/ibmtld` directory. This `Nmakefile` file is, typically, called into execution from the `Nmakefile` located in the `cmd/c1/tp` directory. When this `Nmakefile` file is called into execution with the `install` parameter, `ibmnet` is placed in the `/usr/lib/tp` directory from which the tape daemon calls in execution.
`ibmlib.a`	Contains an archive of modules in relocatable binary format that link into the executable version of `ibmnet`.
`ibmnet.o`	Contains a module in relocatable binary format.

The functionality that is coded within `ibmnet` is only accessible to sites that have obtained license CRIBM/IBM3495 which is required for the IBM 3494 Tape Library Dataserver, and the accompanying Cray FLEXlm key. For assistance with the license, contact your Cray Research service representative.

3.5.4 EMASS autoloader information

The child process started by the tape daemon for an EMASS autoloader is called `esinet` and the vendor-supplied software is called VolServ. The VolServ application typically runs on a Sun platform.

`esinet` is a program in executable binary format for which the building blocks are delivered with the UNICOS release materials. The `cmd/c1/tp/dtdl` directory contains these building blocks.

File	Description
`Nmakefile`	Contains instructions to create esinet out of `esinet.o` and `esilib.a`. It is installed in the `cmd/c1/tp/dtdl` directory. This `Nmakefile` file is, typically, called into execution from the `Nmakefile` located in the `cmd/c1/tp` directory. When this `Nmakefile` file is called into execution with the `install` parameter, `esinet` is placed in the `/usr/lib/tp` directory from which the tape daemon calls into execution.
`esilib.a`	Contains an archive of modules in relocatable binary format that link into the executable version of `esinet`.
`esinet.o`	Contains a module in relocatable binary format.

The functionality which is coded within `esinet` is only accessible to sites that have obtained license CREMS/DTDL and the accompanying Cray FLEXlm key. For assistance with the license, contact your Cray Research service representative.

3.5.5 General installation information

To use an IBM autoloader, an EMASS autoloader, or the UNIX version of the StorageTek autoloader, you must build your system with TCP/IP turned on in the `/etc/config/config.mh` file. That is, `/etc/config/config.mh` must contain the following lines:

```
#define CONFIG_TCP 1
#define CONFIG_RPC 1
```

The UNIX storage server host name must be defined in the local `/etc/hosts` file. For more information, see the `hosts`(5) man page. The UNIX storage

system host name also must be specified in the `server` parameter of the `LOADER` definition in the `/etc/config/text_tapeconfig` file.

For the UNIX version of the StorageTek autoloader, you must set `CSI_UDP_RPCSERVICE` and `CSI_TCP_RPCSERVICE` to `TRUE` in the `/usr/ACSSS/rc.acsss` file of the UNIX storage server host.

It is recommended that you use the installation documentation for the autoloaders at your site to correctly install these products.

3.5.6 Organizing your devices in attended and unattended modes

If the autoloader is the only mechanism your site uses to service mount requests, you may skip the following discussion. However, if the tape daemon processes mount requests in a mixed environment, you must organize the devices to use the devices and loaders in the most efficient manner possible.

A *mixed environment* consists of devices serviced by a manual operator and devices serviced by an autoloader. A volume has a domain associated with it and, as such, has a preferred or best loader to service a mount request. If the domain of a tape cartridge is a tape vault, the best loader is an operator. If the tape cartridge resides in the autoloader's domain (silo), the best loader is the autoloader.

Each tape device belongs to a *device group*, which is a collection of devices with equivalent physical characteristics. Although cartridge devices can have equivalent physical characteristics, you should consider the manner in which the devices will be serviced to determine whether or not they should be grouped.

One of the principal reasons for using an autoloader is that the loader can be run in unattended mode (that is, without an operator). Using the autoloader in this manner means that no imports or exports are considered, and a user-requested tape mount that cannot be satisfied by the autoloader is canceled.

The easiest way to prevent canceled mounts is to assign the autoloader drives to a device group different from the one serviced by manual operators. A user can then determine whether the required device group is available before requesting a tape mount. The only drawback to this method is that the user must be aware of the domain in which the tape resides and, if necessary, make changes to scripts if the domain of the tape changes.

For operations that have 24-hour operator coverage, all tape cartridges can be assigned to one device group, with the operator deciding whether the mount request should be queued or canceled, or whether the volume should be

imported or exported. In this case, the user need not be concerned about the domain of the tape.

3.5.7 Accessing tape cartridges

Another administration issue is the accessibility of tape cartridges in an autoloader. In the past, control of a volume serial number (VSN) was provided by an operator or by security programs on a front-end computer. With an autoloader, control of VSNs does not exist; therefore, with the distributed tape daemon software, any user may request the mounting of any VSN in the domain of the autoloader.

On all Cray Research systems, the released routines reside in the `vsnexit.c` module; you can change the module to suit the particular needs or desires of your site.

3.6 Message daemon and operator interface

The message daemon and its associated operator interface provide mount messages for administrators and operators who are loading and unloading tapes. This subsection provides a brief overview of the daemon and interface.

You must have super-user privileges to start or stop the message daemon.

Only one message daemon can be running at any time. If you attempt to start the message daemon while it is already running, an error message will be returned.

All messages are logged by the message daemon as they are received. The logs are kept in the `msglog.log` log file in the `/usr/spool/msg` directory. The `/etc/newmsglog` shell script saves the last several versions of the log. The versions are called `msglog.log.0`, `msglog.log.1`, and so on, with `msglog.log.0` being the most recent. This script also instructs the message daemon to reopen the log file; it should be run from the `crontab`(1) command.

The message daemon request pipe is located in the `/usr/spool/msg` directory.

Table 1 shows the message daemon commands and the privileges required to access them.

Table 1. Message daemon commands

Command	Privilege	Description
`msgdaemon`(8)	Super user	Starts the message daemon
`msgdstop`(8)	Super user	Stops the message daemon
`oper`(8)	Operator	Operator display; displays messages
`msgi`(1)	All users	Sends informative message to operator
`msgr`(1)	All users	Sends action message to operator

An operator is defined as anyone with a special operator group ID. The default group is `operator`. This group ID can be changed in the `Nmakefile` file in the `/usr/src/cmd/msg` directory. On Trusted UNICOS or MLS systems, the `sysops` category is also necessary.

The operator display provided by the `oper`(8) command can be run from any terminal defined in the `/usr/lib/terminfo` file. It requires at least 80 columns and 24 lines. The three lines at the bottom of the operator display screen are used for input and for running commands that do not display information on the screen. The rest of the screen is used as a refresh display to display messages and to run other display commands.

Configuration file `$HOME/.operrc` lists the commands to be run as refresh displays and those that require full control of the screen. `$HOME` is the user's home directory. If this file does not exist, the default configuration file `/usr/lib/oper.rc` is used.

Commands not listed in the configuration file are assumed to be nondisplay commands.

The following are three of the commands available from the operator display:

Command	Description
`infd`(8)	Displays informative messages
`msgd`(8)	Displays action messages
`rep`(8)	Replies to action messages

Two types of messages appear on the operator interface:

- Informative messages, which are deleted after the operator has seen them. The operator cannot reply to informative messages.
- Action messages, which require a reply from the operator. These are primarily tape mount messages, but they may be other types of messages to which users need responses. An action message is not deleted until either the operator replies to it or the sender cancels it.

Both types of messages are logged by the message daemon.

Tape Troubleshooting [4]

Occasionally you may experience problems with the hardware or the software while running magnetic tapes. If so, there are certain steps you should take to try to clear the user, job, tape drive, or the tape daemon itself, and save the proper information to debug the problem.

This chapter describes the following troubleshooting topics:

- Tape drive or job problems
- Tape daemon problems
- `tpdfixup` utility
- Tracing
- Sample trace analysis
- `errpt`(8) utility
- Sample `errpt`(8) analysis
- `daemon.stderr` file
- `crash`(8) or`crashmk`(8) utility

4.1 Tape drive or job problems

If a tape drive appears to be hung, but the tape daemon is still responding to commands such as `tpstat`(1) and `tpgstat`(8), you can use the `tpfrls`(8) command to clear the user's tape reservation. You can determine the user ID and job ID to use with `tpfrls` from either `tpstat` or `tpgstat`. If this does not work, try the `tpclr`(8) command, with the tape device ID as an operand. If the problem appears to be hardware related, free the user by the preceding methods (check this with the `tpstat`(1) command). Then configure the drive down with the `tpconfig`(8) command, and discuss the problem with the appropriate hardware personnel.

4.2 Tape daemon problems

If the tape daemon (see `tpdaemon`(8)) is hung (no tapes moving, no response from any tape commands), and you must take the tape daemon down, first try

the tpdstop(8) command. If this does not work, or the command hangs, determine the process ID of the tape daemon (by using the ps(1) command), and enter the following command line:

```
kill -2 pid
```

The *pid* argument is the process ID of tpdaemon. If kill -2 does not work, enter the following command line:

```
kill -9 pid
```

If you want to report the preceding or other tape problems (such as the abnormal termination of the tape daemon), it is important that you save the trace files from the tape daemon. These files help to track down a tape problem. The trace files are kept in a directory set up during the initial installation of the tape daemon; see TRACEPFX in the /usr/include/tapereq.h file to find out where these files are kept.

The default installation of the trace files is in the /usr/spool/tape directory. Copy these files as follows:

```
cd directory
cp /usr/spool/tape/trace.* .
```

The *directory* argument is the directory in which you want to keep the trace files. It is also a good idea to create a file or note that explains what the problem was and specifies the devices that were affected: you may also want to keep a copy of the user job that seemed to cause the problem.

Another useful command is the tpbmx(8) command. tpbmx specified with the -d option displays the tape driver's tables for every device. It is recommended that you save a copy of the tpbmx -d output before attempting to execute tpdstop or attempting to terminate the daemon with the kill(1) command.

4.3 tpdfixup utility

The tpdfixup utility collects information pertinent to the online tape subsystem on Cray Research computer systems. A privileged user may run this script when a tape related problem occurs. The information is placed in a separate directory so it can be easily packaged and shipped for offline analysis. For the collected information to be of optimal use, tracing for the tape subsystem should be enabled. For detailed information, please contact the Cray Research Technical Support Center.

Before anything is copied to the information directory, the `tpdfixup` utility attempts to determine whether the tape daemon is in its normal state, and if not, runs a few checks for known hang situations.

The `tpdfixup` utility should be executed to gather information once trouble with the tape daemon is suspected prior to attempting to terminate the tape daemon.

4.4 Tracing

Tracing for the tape subsystem is turned on by default. All child processes created by the tape daemon have tracing enabled. While tracing is a very important tool for debugging tape subsystem problems, it uses additional CPU time. Tracing can be turned on and off by issuing the `tpset`(8) command. To turn tracing off, enter the following command:

```
tpset -T off
```

To turn tracing on, enter the following command:

```
tpset -T on
```

If the stability of the tape subsystem at a site has been established, tape tracing may be an unnecessary overhead. The CPU cycles saved by turning tracing off depends on the mix of jobs submitted, because some tape operations generate more trace information than others.

When tracing is turned off, the tape daemon and its child processes still trace entry to and exit from child processes and abnormal termination of tape processes. Abnormal terminations include those induced by the operator and terminations caused by errors within the tape subsystem. A tape mount request canceled by an operator or interrupted user job is considered an abnormal termination induced by the operator.

The option of turning tracing off for the tape subsystem allows sites running with a stable tape subsystem to substantially reduce the system and user time used by the tape daemon. This gain in system and user time must be weighed with the knowledge that some error information and all trace information will be lost in case of a tape daemon problem. The only way to analyze the problem is to turn tracing on, resubmit the job, and collect traces when the problem reappears.

4.5 Sample trace analysis

To obtain a complete picture of a problem, save trace information as soon as possible after you identify an error situation. You can use the `tpdfixup` utility to aid in the data gathering process.

This utility saves all the pertinent trace files in `/usr/spool/tape` as well as kernel traces through the issuance of `crash`(8) or `crashmk`(8) commands (in particular `tpt` and `tps`). If the tape daemon is not hung, the `display` command output is also saved. When you execute the utility, you are asked to comment on how the system was behaving at the time `tpdfixup` was run.

All of the trace files are circular. For instance, if a particular tape drive is hung, by the time it is noticed the tape daemon trace (`trace.daemon`) has probably been overwritten. However, the drive trace (`trace.bmx###`) and the kernel drive trace should provide some useful information. By default, the drive traces are 409600 bytes in length while the `trace.daemon` file is 10 times that value (the default is 4096000 bytes). This parameter is configurable in the tape configuration file.

Each time a tape daemon routine is entered, tracing for that routine begins. This is done by using the `FUNC` function defined in the `tape.h` file. `RETURN` and `EXIT`, also defined in `tape.h`, indicate when the routine is done.

Within each routine, you can place calls to the trace function to obtain more detailed information. By using this information, you can trace the paths that the software took to perform various tape functions.

When `tpdaemon`(8) forks off its children, (for example, `opentdt` and `readerr`) their trace information is written into the respective tape daemon device traces (`trace.bmx###`). There are also trace files for `avrproc`, `stknet`, `esinet`, and `tcpnet`. By using all of the appropriate traces, you can obtain the entire picture of what was happening when a failure occurred.

4.5.1 Trace information

The following example shows the information you obtain from a trace line.

```
10:59:58 151257598.1241 1450 tpdaemon mounttp function entered
^^^^^^^^ ^^^^^^^^^^^^^^ ^^^^ ^^^^^^^^ ^^^^^^^ ^^^^^^^^^^^^^^^^.......
AAAAAAAA BBBBBBBBBBBBBB CCCC DDDDDDDD EEEEEEE FFFFFFFFFFFFFFFF.......
```

The fields in this line are labeled as follows:

Field	Description
A	References the wall clock time. Having this time available is helpful in relating events in one trace to other traces, `errpt`(8) files, console messages or `daemon.stderr` messages.
B	References the real time clock. You use this time when timing issues are more important. It helps to determine whether the events truly took place in the proper order.
C	References the process number of the main routine. In the `trace.daemon` file, this value will invariably be `tpdaemon`(8); in the `trace.bmx###` files, the value will be the particular child `tpdaemon`(8) forks off to process the request (for example, `opentdt`, or `writeerr`).
D	Identifies the main routine.
E	References the particular routine called by the main routine. In this example, the routine is named `mounttp`.
F	Provides detailed trace information about the entry. This example shows that the `mounttp` function was entered.

4.5.2 Trace example

The following example shows what happens when a user issues an `rsv`(1) command. The listing contains fields `E` and `F` of the trace information from `trace.daemon`.

```
(Start of trace)
getreq function entered
getreq request came from /usr/spool/tape/daemon.request
getreq request X
getreq 000312fe  5472657148000470  0000000000000214  TreqH..p........
getreq 00031300  0000000000000293  0000000000000000  ................
getreq 00031302  0000000000000000  0000000000000000  ................
getreq 00031304  2f7573722f73706f  6f6c2f746170652f  /usr/spool/tape/
getreq 00031306  5c36353972737635  3439353700000000  659rsv54957....
getreq 00031308  0000000000000000  0000000000000000  ................
getreq              ***** same *****
getreq 0003130e  0000000000005b6e  0000000000002e3c  ......[n.......<
getreq 00031310  0000000000005b6e  0000000000000000  ......[n........
getreq 00031312  0000000000000000  0000000000000000  ................
getreq 00031314  2f746d702f6a746d  702e303030363539  /tmp/jtmp.000659
getreq 00031316  612f544150455f52  45515f3635390000  a/TAPE_REQ_659..
```

```
getreq 00031318  0000000000000000  0000000000000000  ................
getreq               ***** same *****
getreq 0003131e  2f636c6f7564792f  75362f6261722f74  /cloudy/u6/bar/t
getreq 00031320  6170652e6d736700  0000000000000000  ape.msg.........
getreq 00031322  0000000000000000  0000000000000000  ................
getreq               ***** same *****
getreq 00031328  6261720000000000  0000000000000000  bar.............
getreq 0003132a  0000000000000001  4341525400000000  ........CART....
getreq 0003132c  0000000000000000  0000000000000001  ................
getreq 0003132e  0000000000000000  0000000000000000  ................
getreq               ***** same *****
getreq 00031342  0000000000000000  0000000000000063  ...............c
getreq 00031344  0000000000000063  0000000000000063  .......c.......c
getreq               ***** same *****
getreq 0003134a  0000000000000000  0000000000000000  ................
getreq 0003134c  0000000000002e3c  0000000000000009  .......<........
getreq 0003134e  00000000000003ef  0000000000000003  ................
getreq 00031350  0000000000003128  00000000000030f6  ......1(......0.
getreq 00031352  0000000000000000  0000000000000000  ................
getreq               ***** same *****
getreq 0003138a  0000000000000000  0000000000000000  ................
getreq getreq returns : code = 1
```

4.5.2.1 Source

The tape daemon checks its request pipes and determines a request is pending. The `getreq` function is entered as shown by the trace entry. While you examine the trace information, you may want to access the `tpdaemon`(8) source. Following the code in `getreq.c` is a trace entry:

```
if (reqhdr.code != TR_TPS) {                    /* don't dump tpstat */
        trace(func,"request came from %s",reqfsp->fn);
        DUMP("request",reqp,reqhdr.size);
}
```

This code traces from where the request came as well as dumping the request. If the request is a `tpstat`(1) command, it is not dumped because the `tpstat`(1) command is issued so often. To determine what the request is, examine the code in word one of the request. In this example, word 1 contains 0000000000000214. The information is dumped in hexadecimal as evidenced by the line `request X`. (A dump in octal would show `request 0`.)

To identify the request, check the `tape.h` file:

```
fir013% grep 214 tape.h
#define TR_RSV          0x214                   /* reserve devices */
```

The request structures for each request are generally contained in the files named `tr` *xxxx.h*. *xxxx* refers to the command issued. To examine the request structure for this example, look in the `trsv.h` file. If a structure does not have its own header (`.h`) file, it is probably located in `tape.h`, the mount tape structure.

Within the `tpdaemon`(1) source is a series of `case` statements. Based on the request code, `tpdaemon`(1) calls the necessary function. In this instance, the request code of x214 corresponds to `TR_RSV`.

```
(from tpdaemon.c)
                case TR_RSV :
                        cfunc = rsvdev;
                        break;

(Trace continued)
rsvdev function entered
gettusr function entered
gettusr gettusr returns : code = 0
addq function entered
addq addq returns : code = 157881
dgpavail function entered
dgpavail dgpavail returns : code = 1
addrsv function entered
gettrsv function entered
gettrsv gettrsv returns : code = 201728
```

The `rsvdev` trace is the next function entered. It calls `gettusr` to determine if the user has already reserved a tape drive. `gettusr` returns a 0 indicating that no reserves are currently assigned to this user. Since a 0 is returned, the following `if` statement is false and the `if block` is bypassed.

```
(from rsvdev.c)
       if (tusrp = gettusr(reqp->rh.jid)) {    /* user found */
```

By looking at the code, you can deduce that this example was run on a system that did have security running because it does not contain any security trace entries.

Many of the tpdaemon(8) subroutines are contained within their own named .c file. Others are contained within various subroutines. If you cannot locate a particular routine, use a grep(1) command on the tpdaemon(1) source to find it.

rsvdev continues on. addq is then entered and returns the queue header pointer to rsvdev.

The dgpavail routine is called to determine if a device is available within the device group requested.

```
(from rsvdev.c)
        FUNC(dgpavail);

        for (i = 0; i < tdth.numdgp; i++) {
                trsvp = tdth.tusrqh.f->trsvp + i;
                if (!strcmp(trsvp->dgn,dgn)) {  /* found */
                        if (num > trsvp->num) {
                                rc = -1;
                        } else {
                                *trsvpa = trsvp;
                                rc = 1;
                        }
                        break;
                }
        }
        RETURN(rc);
```

The value that is returned, 1, indicates that a device is available. A particular return code is neither good nor bad based on its value; you must examine the source to determine the meaning of a code.

```
(from rsvdev.c)
                c = dgpavail(reqp->dgn[i].name,reqp->dgn[i].num,&tdtrsvp);
                if (c > 0) {                            /* available */
                        addrsv(tusrp,reqp->dgn[i].name,reqp->dgn[i].num);
```

Since c is greater than 0, the next block of code is executed. addrsv is called to add to tape reserved. addrsv calls gettrsv to return the address of the trsv structure. The code returned by gettrsv is the decimal address 201728, which converts to 31400 in hexadecimal. The addrsv trace dumps the tusr and trsv structures. The trsv structure is dumped from location x31400:

```
(Trace continued)
addrsv tusr X
addrsv 0003138f  0000000000000000  000000000002bcd4  ................
```

```
addrsv 00031391  2f746d702f6a746d  702e303030363539  /tmp/jtmp.000659
addrsv 00031393  612f544150455f52  45515f3635390000  a/TAPE_REQ_659..
addrsv 00031395  0000000000000000  0000000000000000  ................
addrsv               ***** same *****
addrsv 0003139d  000000000000000d  0000000000000293  ................
addrsv 0003139f  0000000000000000  0000000000000000  ................
addrsv 000313a1  0000000000000000  6261720000000000  ........bar.....
addrsv 000313a3  0000000000000000  0000000000005b6e  ..............[n
addrsv 000313a5  0000000000005b6e  2f636c6f7564792f  ......[n/cloudy/
addrsv 000313a7  75362f6261722f74  6170652e6d736700  u6/bar/tape.msg.
addrsv 000313a9  0000000000000000  0000000000000000  ................
addrsv               ***** same *****
addrsv 000313b1  0000000000000000  0000000000000001  ................
addrsv 000313b3  0000000000000000  0000000000000000  ................
addrsv               ***** same *****
addrsv 000313b7  0000000000031400  0000000000002e3c  ...............<
addrsv 000313b9  0000000000000000  0000000000000000  ................
addrsv               ***** same *****
addrsv 000313bd  0000000000002e3c  0000000000000009  .......<........
addrsv 000313bf  00000000000003ef  0000000000000003  ................
addrsv 000313c1  0000000000003128  00000000000030f6  ......1(......0.
addrsv 000313c3  0000000000000000  0000000000000000  ................
addrsv               ***** same *****
addrsv 000313fb  0000000000000000  0000000000000000  ................
addrsv trsv X
addrsv 00031400  4341525400000000  0000000000000000  CART............
addrsv 00031402  0000000000000001  0000000000000000  ................
addrsv 00031404  5441504500000000  0000000000000000  TAPE............
addrsv 00031406  0000000000000000  0000000000000000  ................
addrsv 00031408  5445535400000000  0000000000000000  TEST............
addrsv 0003140a  0000000000000000  0000000000000000  ................
addrsv 0003140c  3334393000000000  0000000000000000  3490............
addrsv 0003140e  0000000000000000  0000000000000000  ................
addrsv addrsv returns : code = 0
```

The next routine called, `bdmrsv`, sends an `ioctl`(2) system call about the reserve to the kernel.

```
(from rsvdev.c)
       if (ioctl(bmxfs.fd,BDM_RSV,jid) < 0) {
                errmsg(func,ETSYS,TM047,bmxfs.dvn,bmxfs.fn,"ioctl",
                                                    "BDM_RSV",errno);
                RETURN(errno);
```

```
        }
        usrmsg(func,TM000);                     /* tell user about it */

(Trace continued )
bdmrsv function entered
bdmrsv TM000 - tape resource reserved for you
bdmrsv bdmrsv returns : code = 0
```

4.5.2.2 Associated kernel trace entry

The kernel code to process the `ioctl`(2) system call is in `/usr/src/uts/c1/io/tpddem.c`. You can obtained this kernel information by issuing a `tpt tpdemreq` command from within the `tpdaemon`(8) command. These traces are in the oldest-to-latest order; the following is the latest or last trace entry:

```
tpddemct 00000000000000002061 00000010005000000002006 .......1.. .....
```

`tpddemct` is entered as follows:

```
(from /usr/src/uts/c1/io/tpddem.c)
tpddemctl(vp, cmd, arg)
```

The trace is coded as:

```
(from /usr/src/uts/c1/io/tpddem.c)
        TPD_TRACE(io, 'tpddemct', arg, UTPACK(cmd, vp));
```

From the `ioctl`(2) system call in `bdmrsv`, you can equate `vp` to `bmxfs.fd`, `BDM_RSV` to `cmd`, and `jid` to `arg`. Based on the kernel trace entry, 2061 should be the job ID. In this case, 1073 (decimal equivalent of 2061) is the job ID, and 10005 corresponds to the `BDM_RSV` command.

```
(from /usr/src/uts/c1/sys/tpddem.h)
#define TDM_RSV   010005      /* Mark job having device(s) reserved */
```

`rsvdev` then dumps `tusr` and `trsv`, calls `sendrep` to send the reply, and returns with a code of 0 that indicates successful completion.

```
rsvdev tusr X
rsvdev 0003138f  0000000000000000  000000000002bcd4  ................
rsvdev 00031391  2f746d702f6a746d  702e303030363539  /tmp/jtmp.000659
rsvdev 00031393  612f544150455f52  45515f3635390000  a/TAPE_REQ_659..
rsvdev 00031395  0000000000000000  0000000000000000  ................
rsvdev              ***** same *****
rsvdev 0003139d  000000000000000d  0000000000000293  ................
```

```
rsvdev 0003139f  0000000000000000  0000000000000000  ................
rsvdev 000313a1  00599e4eec1ee788  6261720000000000  .Y.N....bar.....
rsvdev 000313a3  0000000000000000  0000000000005b6e  ..............[n
rsvdev 000313a5  0000000000005b6e  2f636c6f7564792f  ......[n/cloudy/
rsvdev 000313a7  75362f6261722f74  6170652e6d736700  u6/bar/tape.msg.
rsvdev 000313a9  0000000000000000  0000000000000000  ................
rsvdev               ***** same *****
rsvdev 000313b1  0000000000000000  0000000000000001  ................
rsvdev 000313b3  0000000000000000  0000000000000000  ................
rsvdev               ***** same *****
rsvdev 000313b7  0000000000031400  0000000000002e3c  ..............<
rsvdev 000313b9  0000000000000000  0000000000000000  ................
rsvdev               ***** same *****
rsvdev 000313bd  0000000000002e3c  0000000000000009  .......<........
rsvdev 000313bf  00000000000003ef  0000000000000003  ................
rsvdev 000313c1  0000000000003128  00000000000030f6  ......1(......0.
rsvdev 000313c3  0000000000000000  0000000000000000  ................
rsvdev               ***** same *****
rsvdev 000313fb  0000000000000000  0000000000000000  ................
rsvdev trsv X
rsvdev 00031400  4341525400000000  0000000000000000  CART............
rsvdev 00031402  0000000000000001  0000000000000000  ................
rsvdev 00031404  5441504500000000  0000000000000000  TAPE............
rsvdev 00031406  0000000000000000  0000000000000000  ................
rsvdev 00031408  5445535400000000  0000000000000000  TEST............
rsvdev 0003140a  0000000000000000  0000000000000000  ................
rsvdev 0003140c  3334393000000000  0000000000000000  3490............
rsvdev 0003140e  0000000000000000  0000000000000000  ................
sendrep function entered
sendrep sendrep returns : code = 0
rsvdev rsvdev returns : code = 0
```

4.6 `errpt`(8) utility

The `errpt`(8) utility processes data collected by the error-logging mechanism (`errdemon`(8)) and generates a report of that data. The default report is a summary of all errors posted in the files specified on the command line. The options apply to all files. If you do not specify any files, `errpt`(8) attempts to use the `/usr/adm/errfile` file.

A summary report notes the options that can limit its completeness, records the time stamped on the earliest and latest errors encountered, and specifies the

total number of errors of one or more error types. The number of times that errpt(8) has difficulty reading input data is included as read errors.

A detailed report contains, in addition to specific error information, all instances in which the error logging process was started and stopped, and the time changes (using the date(1) command) that may have occurred during the interval being processed. A summary of each error type included in the report is appended to a detailed report.

A report can be limited to certain records by the use of options.

For the tape subsystem, the errpt(8) command generates information useful for debugging both hardware and software. For more information, see the errpt(8) man page.

The following example will generate a detailed report about tape devices:

```
errpt -f -d tape
```

4.7 Sample errpt(8) analysis

The errpt(8) analysis available for SCSI protocols is more detailed than that for the block multiplexer (mux) and ESCON protocols. The samples in this section illustrate this difference.

4.7.1 Block mux and ESCON protocols

This analysis deals with errpt(8) tape errors for the block mux and ESCON protocols. Error information is generally logged in /usr/adm/errfile. When these logs are restarted, they are saved as files named errfile # where # is a sequential number starting with and incrementing. The errpt(8) program or the UNICOS olhpa(8) program reads the logs and formats the data. Error messages reported by errpt(8) are created by the bmxereclog routine called from the bmx routines in /usr/src/uts/c1/io.

You can also display these messages on the console by using the bmxconmsg routine. The console messages generally have the following form:

```
ebmx: cart04: unassign, command reject, C040002700000020 0000154400000000
AAAA  BBBBBB  CCCCCCCC  DDDDDDDDDDDDD  EEEEEEEEEEEEEEE EEEEEEEEEEEEEEE
```

The fields in this line are labeled as follows:

Field	Description
A	Indicates the calling program, `ebmx`.
B	Indicates the device on which the error occurred.
C	Shows the `bmx` command that was issued.
D	Shows the resulting error message.
E	Usually records sense bytes 0 through 15. Verify by checking the specific error message in `ebmx.c`.

For aid in breaking down the sense bytes, see the appropriate IBM documentation.

The following command produced the sample `errpt`(8) record:

```
errpt -d tape -s 11011400 -e 11011500
```

The `-s` and `-e` parameters refer to the starting and ending times that were used. They are in the `mmddhhmm` format. The `-d` parameter indicates that `errpt`(8) should report on tape errors.

```
Tue Nov  1 14:53:53 1994
        Tape Error record Cluster 0 IOS 1 Device 150


        Volume:    Owner: 0   Command:

        (CART)  Error type: Drive assigned elsewhere  Final status: UNRECOVERED


        Initial channel: 036  Initial control unit: 013  Initial device: 000

        Final channel  : 000  Final control unit  : 000  Final device  : 000

        Request code: 0x9a  Response code: 0x0e

        Channel command: Assign

        Initial status (erpa): 0x045  Extended status: 0x2002

        Initial device status : 0x02  Final device status : 0x00

        Block: 0  Density: 0  Retry count: 00000

        Sense bytes: (hexadecimal)
```

```
00   -      41          40          80          45
04   -      00          00          00          20
08   -      01          40          33          e4
12   -      00          00          00          00
16   -      00          00          00          70
20   -      00          00          00          00
24   -      f6          80          34          72
28   -      20          50          00          00
```

This sample is a relatively straightforward `errpt`(8) record. If a tape job were involved, the volume, owner, and command fields would contain relevant information. However, the error type field indicates that the drive was assigned elsewhere with a final status of unrecovered.

The channel is octal 36, the control unit is octal 13, and the drive ID (initial device) is 0. You can verify this information in the tape configuration file:

```
{
        CONTROL_UNIT
                protocol = STREAMING ,
                status = UP ,
                path = ((036, 11))
        DEVICE
                name = 150 ,
                device_group_name = CART ,
                id = 00 ,
                type = 3480 ,
                status = DOWN ,
                loader = Operator
```

The request code of `x9a` indicates a command list, and the response code of `x0e` is a sequencer detected error. These commands are in the `/usr/include/sys/epackt.h` file under request codes to the IOS and IOS response codes.

```
/*
 * Define request codes to ios
 */

#define TCommandList                    0232

/*
 * IOS response codes
```

```
 */

#define RUnitCheck                          016
```

The channel command is `assign`. The ERPA code of `x045` can be located in the `/usr/include/sys/erec.h` file.

```
#define         T3480_DAE       0x45    /* Drive assigned elsewhere */
```

4.7.2 SCSI protocols

The sample shows the additional information that is available for SCSI protocols.

```
Tue Aug  6 15:48:53 1996
        Tape Error record Cluster 3 IOS 2 Device s9490s0


        Volume:    Owner: 40   Command:

        (CART)  Error type: Read data check  Final status: UNRECOVERED


        Initial channel: 002  Initial control unit: 002  Initial device: 000

        Final channel  : 000  Final control unit  : 000  Final device  : 000

        Request code: 0x9a  Response code: 0x0e

        Channel command: Load display

        Initial status (erpa): 0x023  Extended status: 0x400e

        Initial device status : 0x0e  Final device status : 0x00

        Block: 0  Density: 0  Retry count: 00000

        Sense bytes: (hexadecimal)
                 00   -      48          40          00          23
                 04   -      00          00          00          00
                 08   -      00          00          00          00
                 12   -      00          00          00          00
                 16   -      00          00          00          00
```

```
          20   -      00          00          00          00
          24   -      00          00          00          00
          28   -      00          00          00          00
          32   -      00          00          03          00
          36   -      00          00          00          00
          40   -      00          00          00          00
          44   -      11          01          00          00
          48   -      00          00          00          00
          52   -      00          00          00          00
          56   -      00          00          00          00
          60   -      00          00          00          00

SCSI Sense Byte 2 bits 3 - 0: 0x3(Medium Error)

SCSI Sense Byte 2 bits 7 - 5: 0x0

SCSI Sense Bytes 12/13: 0x1101(Read Retries Exhausted)

SCSI Sense bytes: (hexadecimal)
          00   -      00          00          03          00
          04   -      00          00          00          00
          08   -      00          00          00          00
          12   -      11          01          00          00
          16   -      00          00          00          00
          20   -      00          00          00          00
          24   -      00          00          00          00
          28   -      00          00          00          00
```

4.8 `daemon.stderr` file

The `/usr/spool/tape/daemon.stderr` file contains all tape daemon error messages. Therefore, this file contains debug information that helps diagnose errors. This file, along with the output from `errpt`(8), is useful for administrators when working on drive problems. It is also useful for debugging tape daemon problems when sent with other tape daemon trace files to Cray Research for offline analysis.

4.9 `crash`(8) or `crashmk`(8) utility

The `crash`(8) or `crashmk`(8) utility can help you discover and correct tape subsystem problems. This interactive utility can examine an operating system

core image. It has facilities for interpreting and formatting the various control structures in the system and certain miscellaneous functions that are useful when examining a dump file.

The *core_filename* argument specifies where the system image can be found. The default value of *core_filename* is `/dev/mem`, which lets you use the `crash`(8) or `crashmk`(8) utility without an operand to examine an active system. If you specify the system image file, it is assumed to be a system core dump and the default process is set to that of the process active in the kernel at the time of the crash. This is determined by a value stored in a fixed location by the dump mechanism.

The following `crash`(8) or `crashmk`(8) commands are useful for tape problem solving:

`tpt` [*device1*][*device2*]...

Prints kernel level tape device traces. `tpt` called without any arguments prints out a table containing the device name (as seen in the `tpstat`(1) display); index (physical device name); and the start, middle, and end trace pointers for each device in the tape table. `tpt` called with a device name prints out traces for that device.

On UNICOS systems, `tpt` called with a dash (`-`) instead of *device1* dumps out traces for all tape devices in the system.

For more information concerning the `tpt` command on UNICOS/mk systems, use `help tpt` from within the `crashmk`(8) utility.

`tps` [*device1*][*device2*]...

(UNICOS systems only) Prints tape device structures. `tps` called without any arguments prints out tape I/O structures for all tape devices in the system. `tps` called with a device name prints out the tape structures associated with that device. `tps` called with a dash (`-`) instead of *device1* prints out tape structures for all tape devices in the system.

Man Pages [A]

This chapter lists the administration commands and utilities associated with the tape subsystem. Use the `man`(1) command to access the individual man pages. The following four categories of man pages exist:

- Devices (special files)

  `tape`(4)

  `tpddem`(4)

- File formats

  `tapereq`(5)

  `tapetrace`(5)

  `text_tapeconfig`(5)

- Miscellaneous topics

  `tape`(7)

- Administration commands

  `tpapm`(8)

  `tpbmx`(8)

  `tpclr`(8)

  `tpconf`(8)

  `tpconfig`(8)

  `tpcore`(8)

  `tpdaemon`(8)

  `tpdev`(8)

  `tpdstop`(8)

  `tpformat`(8) (UNICOS systems only)

  `tpfrls`(8)

  `tpgstat`(8)

  `tpinit`(8)

`tplabel`(8)

`tpmls`(8)

`tpmql`(8)

`tpscr`(8)

`tpset`(8)

`tpu`(8)

`xtpldr`(8)

Index

S

T

U