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.

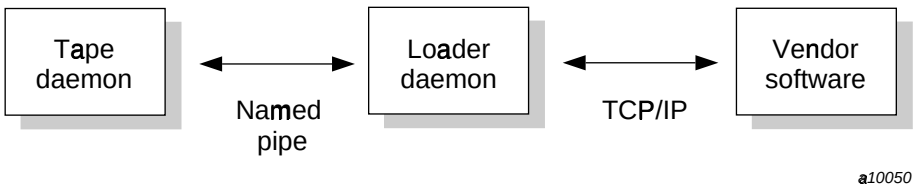

Figure 1. UNIX autoloader communication

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.