TMF 6 Administrator Guide for SGI® InfiniteStorage

007–5534–003

New Features in this Guide

This revision contains minor corrections, updated titles, and updated corporate information.

Record of Revision

Version	Description
001	March 2009 Supports TMF 4.1 in ISSP 1.6
002	June 2010 Supports TMF 5.1 in ISSP 2.1
003	November 2013 Supports TMF 6.1 in ISSP 3.1

Contents

Figures

Tables

Procedures

About This Guide

This guide tells you how to configure, administer, and troubleshoot the Tape Management Facility (TMF) mounting service for the Data Migration Facility (DMF).

Related Publications

See the following

- *DMF 6 Administrator Guide for SGI InfiniteStorage*
- *OpenVault Administrator Guide for SGI InfiniteStorage*

TMF provides the following man pages:

- User commands:

  `msgr`(1)
  `tmcatalog`(1)
  `tmlist`(1)
  `tmmnt`(1)
  `tmrls`(1)
  `tmrst`(1)
  `tmrsv`(1)
  `tmstat`(1)

- Files:

  `tmf.config`(5)
  `tmfctl`(5)
  `tmftrace`(5)

- Administrator commands:

msgd(8)	tmcollect(8)	tmlabel(8)
msgdaemon(8)	tmconf(8)	tmmls(8)
msgdstop(8)	tmconfig(8)	tmmql(8)
msgrep(8)	tmdaemon(8)	tmset(8)
oper(8)	tmfrls(8)	tmstop(8)
tmclr(8)	tmgstat(8)	tmunld(8)

Also see the following Linux man pages:

chkconfig(8)
kill(1)
ps(1)
ts(7)

Obtaining Publications

You can obtain SGI documentation as follows:

- See the SGI Technical Publications Library at http://docs.sgi.com. Various formats are available. This library contains the most recent and most comprehensive set of online books, man pages, and other information.
- You can view man pages by typing `man` *title* at a command line.
- The `/docs` directory on the ISSP DVD or in the Supportfolio download directory contains the following:
  - The ISSP release note: `/docs/README.txt`
  - DMF release notes: `/docs/README_DMF.txt`
  - The manuals provided with ISSP
  - A complete list of the packages and their location on the media: `/docs/RPMS.txt`
  - The packages and their respective licenses: `/docs/PACKAGE_LICENSES.txt`
- The ISSP release notes and manuals are installed on the system as part of the `sgi-isspdocs` RPM into the following location:

  `/usr/share/doc/packages/sgi-issp-`*ISSPVERSION*`/`*TITLE*

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.
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.

Reader Comments

If you have comments about the technical accuracy, content, or organization of this publication, contact SGI. Be sure to include the title and document number of the publication with your comments. (Online, the document number is located in the front matter of the publication. In printed publications, the document number is located at the bottom of each page.)

You can contact SGI in either of the following ways:

- Send e-mail to the following address:

  techpubs@sgi.com

- Contact your customer service representative and ask that an incident be filed in the SGI incident tracking system:

  http://www.sgi.com/support/supportcenters.html

SGI values your comments and will respond to them promptly.

TMF Configuration

This chapter contains the basic information that you need to configure the TMF configuration file (`/etc/tmf/tmf.config`). It presents a complete example file followed by detailed explanations of the statements and key parameters in the example:

Note: You must configure `/etc/tmf/tmf.config` with values appropriate to your site before starting TMF. You can update the file with any text editor. For a complete description of all possible parameters, see the `tmf.config`(5) man page.

Example TMF Configuration File

The following example TMF configuration file begins with a comment (the title of the file) preceded by the number sign character (#).

```
#
#       TAPE MANAGEMENT FACILITY CONFIGURATION FILE
#
#
#

LOADER
      name = operator ,
      type = OPERATOR ,
      status = UP ,
      mode = ATTENDED ,
      message_path_to_loader = MSGDAEMON ,
      server = localhost ,
      queue_time = 0 ,
      verify_non_label_vsn = YES ,
```

```
      message_route_masks = (MSGD) ,
      loader_ring_status = ALERT

LOADER
      name = wolfy ,
      type = STKACS ,
      status = DOWN ,
      mode = ATTENDED ,
      message_path_to_loader = NETWORK ,
      server = wolfcreek ,
      queue_time = 15 ,
      verify_non_label_vsn = NO ,
      message_route_masks = (MSGD) ,
      loader_ring_status = IGNORE

LOADER
      name = panther ,
      type = STKACS ,
      status = DOWN ,
      mode = ATTENDED ,
      message_path_to_loader = NETWORK ,
      server = stk9710 ,
      queue_time = 15 ,
      verify_non_label_vsn = NO ,
      message_route_masks = (MSGD) ,
      loader_ring_status = IGNORE

LOADER
      name = tmfov ,
      type = OPENVAULT ,
      server = armadillo ,
      status = down ,
      mode = ATTENDED ,
      message_path_to_loader = NETWORK ,
      ov_tmf_application_name = tmf,
      queue_time = 15 ,
      verify_non_label_vsn = NO ,
      message_route_masks = (MSGD) ,
      loader_ring_status = IGNORE
```

```
DEVICE_GROUP
      name = CART

DEVICE_GROUP
      name = DLT

DEVICE_GROUP
      name = STK9490

AUTOCONFIG
{
      DEVICE
                name   = t1 ,
                device_group_name = CART ,
                file   = /dev/ts/pci0001:00:03.0/scsi/target1/lun0 ,
                status = DOWN ,
                loader = wolfy ,
                vendor_address = (0,0,1,1)
      DEVICE
                name   = t4 ,
                device_group_name = CART ,
                file   = /dev/ts/pci0001:00:03.0/scsi/target2/lun0 ,
                status = DOWN ,
                loader = wolfy ,
                vendor_address = (0,0,1,0)
      DEVICE
                name   = dlt2 ,
                device_group_name = DLT ,
                file   = /dev/ts/pci0001:00:03.0/scsi/target3/lun0 ,
                status = DOWN ,
                loader = panther ,
                vendor_address = (1,0,2,0)
      DEVICE
                name   = dlt3 ,
                device_group_name = DLT ,
                file   = /dev/ts/pci0001:00:03.0/scsi/target4/lun0 ,
                status = DOWN ,
                loader = panther ,
                vendor_address = (1,0,2,1)
      DEVICE
                name   = s9490s4 ,
```

```
                device_group_name = STK9490 ,
                file   = /dev/ts/pci0001:00:03.0/scsi/target6/lun0 ,
                status = down ,
                vendor_address = (0,0,1,0),
                loader = tmfov
      DEVICE
                name   = s9490s1 ,
                device_group_name = STK9490 ,
                file   = /dev/ts/pci0001:00:03.0/scsi/target7/lun0 ,
                status = down ,
                vendor_address = (0,0,1,1),
                loader = tmfov
}
OPTIONS
trace_directory                        = /var/spool/tmf/trace ,
trace_file_size                        = 409600 ,
trace_state                            = ON
```

Statement Order

The configuration file consists of statements. A *statement* consists of a statement name followed by a list of parameters or other statements. There are at least four statements in a TMF configuration file, one of which also consists of statements. The statements must appear in the following order:

1. LOADER statements (one per loader).
2. DEVICE_GROUP statements (one per device group).
3. AUTOCONFIG statement (one per system).

   The AUTOCONFIG statement consists of DEVICE statements. DEVICE statements (one per device) define devices that TMF will control and that are automatically configured during the system boot.
4. OPTIONS statement (one per system).

Statement Syntax

The following syntax rules apply to the TMF statements:

- Comments begin with the # character.
- The statement name and its parameters are separated by one or more white spaces (blank, tab, or newline characters).
- Adjacent parameters are separated by a comma.
- The end of the parameter list is indicated by the absence of a comma.
- Adjacent statements are separated by one or more white spaces.

The following syntax rules apply to keyword parameters:

- The keyword is separated from its value by the equal sign (=).
- The value of a keyword may consist of keywords, numbers, character strings, and lists of keywords, numbers, and character strings.
- If the value of a keyword is a list, then the list is enclosed within left and right parentheses. Adjacent elements of a list are separated by a comma. If the list consists of one element, you do not have to enclose it in parentheses. The elements of a list may be lists.
- Numbers may be specified in decimal, octal, and hexadecimal formats. These formats are the same as those used in the C programming language:
  - Decimal: the first digit is not `0` (for example, `1372`)
  - Octal: the first digit is `0` (for example, `0563`)
  - Hexadecimal: the first 2 characters are either `0x` or `0X` (for example, `0xf2`)
- Character strings are series of characters. If any white space or any of the following special characters is needed in the string, then the string must be enclosed within a pair of double quotation marks (`"`):

  ```
  "
  #
  =
  {
  }
  (
  )
  ’
  \
  ```

Within a pair of double quotation marks, the sequence of characters \ *x*, where *x* is any character, will be replaced by *x*. This is the only way that a " or a \ character may be specified in a quoted string.

- Comments may appear between any symbols described above.

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

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

The following are different:

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

Example Statement Parameter Details

The following sections explain key elements for each of the example statements shown in "Example TMF Configuration File" on page 1:

- "LOADER Statement Parameters" on page 6
- "DEVICE_GROUP Statement Parameters" on page 8
- "AUTOCONFIG Statement Parameters" on page 8
- "DEVICE Statement Parameters" on page 9
- "OPTIONS Statement Parameters" on page 10

LOADER Statement Parameters

"Example TMF Configuration File" on page 1 contains four LOADER statements; these represent the five loaders that are available on this system. Each LOADER statement is composed of the parameters needed to describe a specified loader. For example, the first LOADER statement has the following parameters:

```
LOADER
      name = operator ,
      type = OPERATOR ,
```

```
status = UP ,
mode = ATTENDED ,
message_path_to_loader = MSGDAEMON ,
server = localhost ,
server_reply_wait_time = 300,
queue_time = 0 ,
verify_non_label_vsn = YES ,
message_route_masks = (MSGD) ,
loader_ring_status = ALERT
```

The parameters specify the following information for this loader:

- The loader name is `operator` and it is of type `OPERATOR`. It will be up (running and waiting for tape requests) when TMF is started and be attended by a human operator (TMF will prompt the human operator for intervention).
- The `/usr/sbin/msgdaemon` message daemon (rather than TCP/IP protocol) will send messages to the servicing loader.
- The server name is `localhost`
- The amount of time that the `LOADER` process waits for a server response before declaring a timeout condition is 300 seconds.
- The system will queue a request and wait for the best loader to become available for up to 24 hours. (A value of `0` for `queue_time` indicates that TMF should wait up to 24 hours; a nonzero value specifies the number of seconds to wait.)
- Volume serial numbers (VSNs) without labels must be verified.
- Mount request messages are routed to the message daemon. The loader is alerted to the ring status whenever a tape is mounted.

Note: It may be necessary to specify an alternate network name for the `LOADER` statements that represent network libraries (using a `message_path_to_loader` value of `NETWORK`). If a network library is not connected to the host primary network, you must specify the path with the `return_host` parameter so that the library can return responses to TMF. The `return_host` parameter is used only if it is set; there is no default.

DEVICE_GROUP Statement Parameters

"Example TMF Configuration File" on page 1 contains three DEVICE_GROUP statements, one for each of the system's device groups (one each for CART, DLT, and STK9490). The first DEVICE_GROUP statement defines the CART device group, which supports the automatic volume feature; the other device groups show that they do not support the automatic volume feature (the default value for the avr parameter is NO, therefore it does not need to be specified and its absence implies a value of NO):

```
DEVICE_GROUP
      name = CART

DEVICE_GROUP
      name = DLT

DEVICE_GROUP
      name = STK9490
```

AUTOCONFIG Statement Parameters

The AUTOCONFIG statement in "Example TMF Configuration File" on page 1 is made up of six DEVICE statements (enclosed within curly brackets), one for each device in the system (t1, t4, dlt2, dlt3, s9490s4, and s9490s1):

```
AUTOCONFIG
{
      DEVICE
                name   = t1 ,
...
      DEVICE
                name   = t4 ,
...
      DEVICE
                name   = dlt2 ,
...
      DEVICE
                name   = dlt3 ,
...
      DEVICE
                name   = s9490s4 ,
...
```

```
      DEVICE
                name   = s9490s1 ,
...
}
```

"DEVICE Statement Parameters" on page 9 provides further details.

DEVICE Statement Parameters

A DEVICE statement identifies the tape devices that are available on the system on which TMF is running. The series of DEVICE statements compose the AUTOCONFIG statement (see "AUTOCONFIG Statement Parameters" on page 8).

For example, in the first DEVICE statement shown in "Example TMF Configuration File" on page 1:

```
      DEVICE
                name   = t1 ,
                device_group_name = CART ,
                file   = /dev/ts/pci0001:00:03.0/scsi/target1/lun0 ,
                status = DOWN ,
                loader = wolfy ,
                vendor_address = (0,0,1,1)
```

The parameters specify the following information for this loader:

- The tape device name is t1.
- This device is a member of the CART device group, which is specified by the first DEVICE_GROUP statement (see "DEVICE_GROUP Statement Parameters" on page 8).
- The pathname to the device-specific file is /dev/ts/pci0001:00:03.0/scsi/target1/lun0.

  Note: For the actual location of the device files on your system, see the ts(7) man page.

- When TMF is started, the initial status of the device is down.
- The loader name is wolfy, and it is defined in the second LOADER statement in "Example TMF Configuration File" on page 1.

- The vendor address of the drive in the library is `(0,0,1,1)`.

`OPTIONS` Statement Parameters

The `OPTIONS` statement shows the values that TMF uses for the options. For a complete description the available options, see the `tmf.config`(5) man page.

In "Example TMF Configuration File" on page 1, the defaults are used for all options except the following:

```
trace_directory                         = /var/spool/tmf/trace ,
trace_file_size                         = 409600 ,
trace_state                             = ON ,
```

The above values indicate the following:

- TMF will log debug information to files in `/var/spool/tmf/trace`
- The log files will wrap when they reach a size of 409600 bytes
- Trace messages are enabled

TMF Administration

This chapter describes the following TMF administration topics:

Tape Libraries

This section describes how TMF interacts with the tape-library software and also covers some high-level configuration information:

Communication

TMF always communicates with the tape loader via an intermediate software system that is provided by the library vendor. TMF supports the following:

- Oracle libraries using the ACSLS software interfacet. TMF communicates with the ACSLS software via a child process called `stknet`, which TMF starts after the library is configured up (*up* means that it is running and waiting for tape requests). Check with your StorageTek representative to validate the values of `CSI_UDP_RPCSERVICE` and `CSI_TCP_RPCSERVICE`.

- IBM libraries using the controlled path service (CPS) software interface. TMF communicates with the CPS software via a child process called `ibmnet`, which TMF starts after the library is configured up.

ACSLS and CPS receive requests from TMF and pass them on to the actual tape libraries for processing. They also send responses back to TMF for any given action.

Figure 2-1 shows the software and hardware configuration between the local host and the Oracle and IBM libraries.

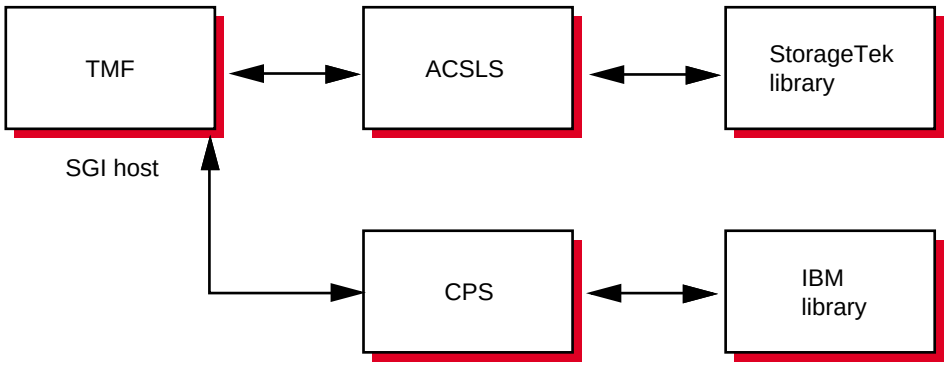

Figure 2-1 Library Communication

Organizing Your Devices in Attended and Unattended Modes

A *mixed environment* consists of devices serviced by a manual operator (*attended mode*) and devices serviced by a library (*unattended mode*). If TMF services mount requests in a mixed environment, you must organize the devices to use both devices and loaders in the most efficient manner possible.

A volume has a domain associated with it and a preferred loader to service a mount request. If the domain of a tape cartridge is a tape vault, the preferred loader is an operator. If the tape cartridge resides in the library's domain (silo), the preferred loader is the library.

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

The easiest way to prevent canceled mounts is to assign the library 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 must 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.

Accessing Tape Cartridges

Another administration issue is the accessibility of tape cartridges in a library. 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 a library, control of VSNs does not exist; therefore, with the distributed TMF software, any user may request the mounting of any VSN in the domain of the library.

OpenVault as a TMF Loader

You can use the OpenVault storage library management facility as a TMF loader. You can configure it on your local host or on a remote host. Figure 2-2 shows OpenVault on the same host as TMF and Figure 2-2 shows OpenVault and TMF on different hosts.

OpenVault supports a wide range of removable media libraries as well as a variety of drives associated with these libraries. The checklists in "Checklists for Using OpenVault with TMF" on page 15 provide information about using OpenVault with TMF. For detailed information about using OpenVault, see the *OpenVault Administrator Guide for SGI InfiniteStorage*.

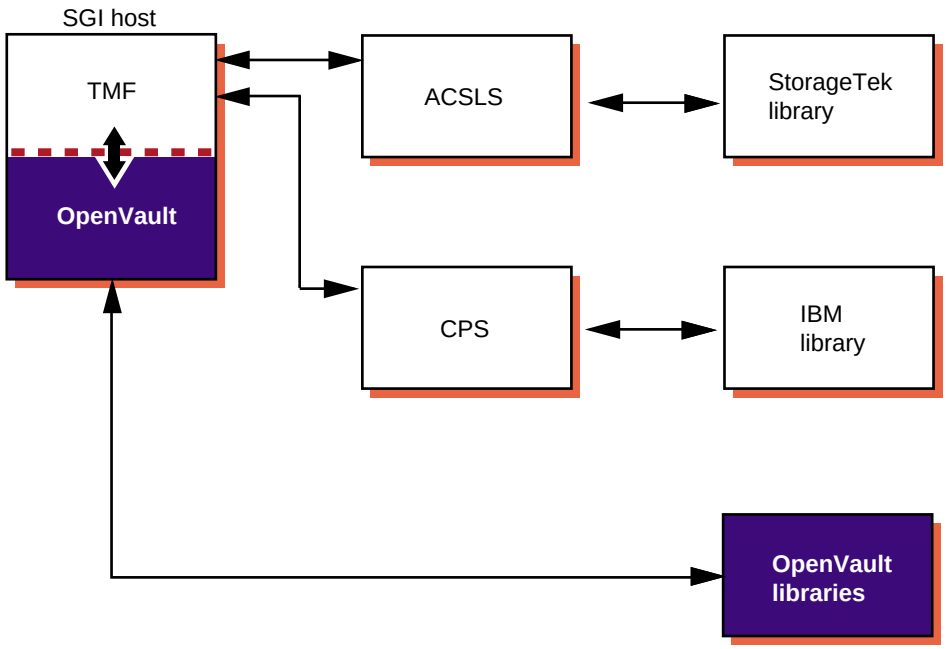

Figure 2-2 OpenVault on the Local Host

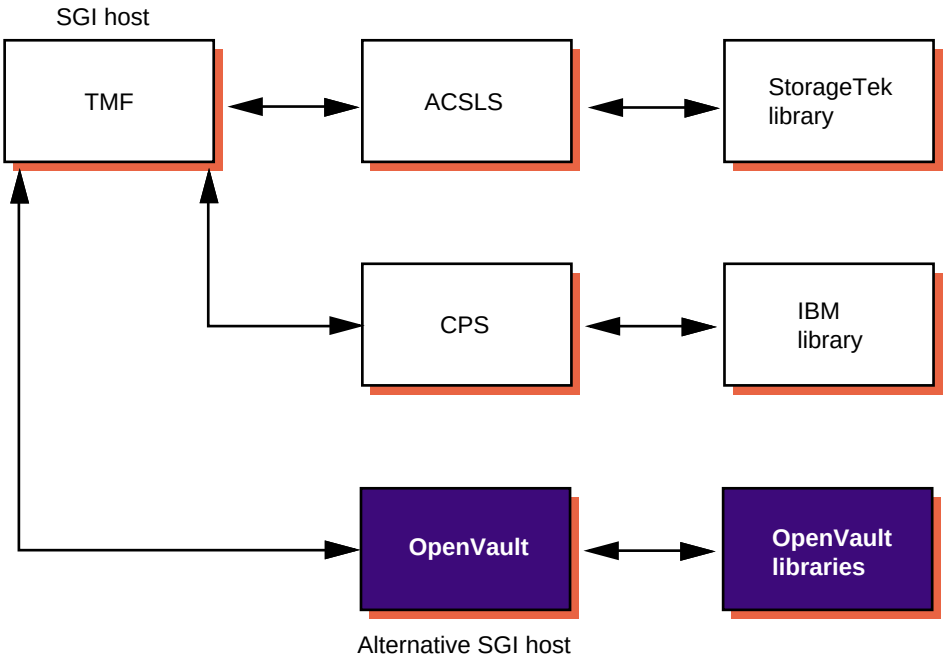

Figure 2-3 OpenVault on a Remote Host

Checklists for Using OpenVault with TMF

The following terms are used in this section:

Term	Description
ovserver_host_name	The name of the host on which the OpenVault server is running. When configuring TMF and OpenVault, use the `hostname` command on the OpenVault server host to obtain this value.
tmf_host_name	The name of the host on which TMF will run. Use the `hostname` command on the TMF host to obtain this value.
tmf_node_name	The node name of the host on which TMF will run. This may or may not be the same as the TMF

hostname. Use the `uname -n` command on the TMF host to obtain this value.

tmf_application_name
: The OpenVault application name to be used by TMF. If you do not specify a value in the TMF configuration file, TMF uses the default name `tmf`.

tmf_instance_name
: The OpenVault instance name to be used by this instance of TMF. If you do not specify a value in the TMF configuration file, TMF uses the following default name:

  tmf_node_name`.`*tmf_application_name*

tmf_key
: The TMF security key to be used for TMF. This alphanumeric string is used as a password by TMF to secure the connection with the OpenVault server. If you want to enable security, you must create and configure a *tmf_keyfile* containing the *tmf_key* that you want to use. If you do not specify a *tmf_keyfile* in the TMF configuration file, TMF uses `none` as the *tmf_key*, which means that security checking between TMF and the OpenVault server is disabled.

tmf_keyfile
: The OpenVault keyfile to be used by TMF when communicating with the OpenVault server. The *tmf_keyfile* contains the private security key *tmf_key* that TMF uses to establish an authorized connection with the OpenVault server. This file is required only if you decide that you want to enable security checking between TMF and the OpenVault server.

  When the key file is created, do the following:

  - Use the `chown` command to set the user ownership to `root`
  - Use the `chgrp` command to set its group ownership to `sys`
  - Use the `chmod` command to set its files permissions to `0500`

The key file should contain a single line consisting of 5 blank or tab-separated fields in the following format:

ovserver_host_name *tmf_application_name* *tmf_instance_name* `CAPI` *tmf_key*

The *ovserver_host_name*, *tmf_application_name*, *tmf_instance_name* values in the *tmf_keyfile* must match the values specified in the TMF configuration file (or the TMF default values, if you do not specify them in the TMF configuration file). Mismatches between the TMF configuration file and the *tmf_keyfile* prevent TMF from contacting the OpenVault server.

Procedure 2-1 and Procedure 2-2, page 22 list the steps you need to take before you use TMF with OpenVault.

OpenVault Checklist

Procedure 2-1 OpenVault Checklist

Configure the drives and libraries in OpenVault so that TMF can use them. The following steps ensure that the TMF and OpenVault counterparts match.

1. Using the OpenVault configuration command, define a list of OpenVault drives and libraries. The configuration command is `/usr/sbin/ov_admin`.

Note: TMF devices are defined subsequently. TMF device names **must** match the drive names used by OpenVault. TMF supports device names up to 8 characters, so you should define OpenVault drive names consisting of 1 to 8 characters.

TMF devices are grouped as follows:

- In TMF, every drive belongs to a device group
- In OpenVault, every drive belongs to a drive group
- For every TMF device group to be managed by OpenVault, there must be a matching OpenVault drive group

2. Create an OpenVault application name so that a group of tapes, defined as a cartridge group, can later be used by the `tmf` application. For example, where `tmf` is the application name:

   ```
   # ov_app -c tmf
   Created Application: tmf
   ```

   In OpenVault, only one application can be assigned to a cartridge. OpenVault mounts a cartridge only if the request comes from the assigned application. Cartridges have the following characteristics:

   - In OpenVault, a cartridge is a physical cartridge (also called a *physical tape* in TMF)
   - Each cartridge is assigned to a cartridge group
   - Each cartridge group is assigned to an application (a client)
   - A cartridge is identified by its physical cartridge label (PCL), which is used to identify a cartridge in an OpenVault loader library

3. Create any new drive groups and/or cartridge groups, according to the following steps. The default group names are `drives` and `carts`, respectively.

   a. Use the following OpenVault commands to list the drive groups and cartridge groups:

   ```
   # ov_drivegroup
   group                unload time
   drives               60

   # ov_cartgroup
   group                group prio
   carts                1000
   ```

   b. Create a new drive group appropriate for your drive types. Keep in mind that this is the same name you will give to the device group you define in TMF. The following example uses `DLT8000` as a drive group name:

   ```
   # ov_drivegroup -c DLT8000
   created drive group: DLT8000
   ```

   To simplify things in the preceding example, the default cartridge group, `carts` is used. However, if you choose to create a new cartridge group, you can do so by using the following command:

   ```
   # ov_cartgroup -c new_cartgroup_name
   ```

   c. Verify that the new drive group (and any new cartridge groups) have been created, as follows:

   ```
   # ov_drivegroup
   group                unload time
   DLT8000              60
   drives               60

   # ov_cartgroup
   group                group prio
   carts                1000
   ```

4. Allow the OpenVault application name, `tmf`, to use the cartridge groups and drive groups that will be used by TMF (defined in step 3).

In OpenVault, more than one application can be assigned to a drive group; OpenVault uses a drive within a drive group only if the request comes from an assigned application. OpenVault drives have the following characteristics:

- In OpenVault, a drive is a TMF device
- Each drive is assigned to a drive group
- Each drive group is assigned to one or more applications (clients)

Use the following steps for all drive and cartridge groups:

a. List applications (for all drive groups and cartridge groups).

   Use the following commands to list the application information for all drive groups and cartridge groups. The default OpenVault application name is `ov_umsh`.

Note: The drive groups (and cartridge group) that were created in step 3 do not appear here. This is because they have not yet been given an application to use.

```
# ov_drivegroup -s -A '.*'
application          group              group app prio  unload time
ov_umsh              drives             1000            60

# ov_cartgroup -s -A '.*'
application          group              group app prio
ov_umsh              carts              1000
```

b. Add TMF applications. Use the following commands with the `-a` option to add the default TMF application name, `tmf`, to any drive groups and cartridge groups that will be used by TMF:

```
# ov_drivegroup -a -G DLT8000 -A tmf
Drive-group-application creation:
       Application: tmf
       Group: DLT8000

# ov_cartgroup -a -G carts -A tmf
Cartridge-group-application creation:
       Application: tmf
       Group: carts
```

c. Recheck: List applications (for all drive groups and cartridge groups), as follows:

```
# ov_drivegroup -s -A '.*'
application          group               group app prio  unload time
ov_umsh              drives              1000            60
tmf                  DLT8000             1000            60

# ov_cartgroup -s -A '.*'
application          group               group app prio
ov_umsh              carts               1000
tmf                  carts               1000
```

5. Create a group of physical cartridges, assigned to the default cartridge group, `carts`, and to be used by the `tmf` application. The fields in the following example are as follows:

   Cartridge PCL name
   OpenVault cartridge type
   Volume name (should match the PCL name)
   Application name

```
# ov_import -g carts
DLT014 DLTIV DLT014 tmf
DLT015 DLTIV DLT015 tmf
<ctrl-d>
#
```

   You can use the following command to list all of the PCLs/volumes used by the `tmf` application:

```
# ov_lscarts -A tmf
DLT014         DLT015
```

   Or, for a more extensive look:

```
# ov_lscarts -l -A tmf
PCL        cart type  owner           state      part     volume
DLT014     DLTIV      tmf             ok         PART 1   DLT014
DLT015     DLTIV      tmf             ok         PART 1   DLT015
```

6. Make sure that the following line is in the core_keys file. The core_keys file is /var/opt/openvault/server/config/core_keys.

   tmf_host_name tmf_application_name tmf_instance_name CAPI *tmf_key*

   For example, if armadillo is the host that TMF is running on (*tmf_host_name*), the TMF application name (*tmf_application_name*) is tmf, the TMF instance name (*tmf_instance_name*) is armadillo.tmf, the language is CAPI, and the security key (*tmf_key*) is not used (none), you would enter the following line in the core_keys file:

   ```
   armadillo  tmf   armadillo.tmf    CAPI  none
   ```

   In the OpenVault documentation, the terminology may differ: the *tmf_keyfile* file is the key authorization file, and the TMF application name (*tmf_application_name*) is the client.

7. To collect debugging information in the OVLOG file, enter the following command:

   # **ov_msg -s -t core -m debug**

   The OVLOG file is /var/opt/openvault/server/logs/OVLOG.*YYYYMMDD*, where *YYYYMMDD* represents the year (*YYYY*), month (*MM*), and day (*DD*) that the file was created..

   When you have finished debugging the OVLOG file, you can use the following command to reset the messages back to the information level:

   # **ov_msg -s -t core -m information**

TMF Checklist

Procedure 2-2 TMF Checklist

Modify the tmf.config file to support the OpenVault loader, device groups, and devices. For information on the tmf.config file, including a complete example file, see the tmf.config(5) man page.

1. Define an OpenVault LOADER statement, as follows:

   a. Define the type to be OPENVAULT:

      ```
      type = OPENVAULT
      ```

b. Specify where the OpenVault server is listening by entering the name of the host:

   `server =` *ovserver_host_name*

c. Either use the TMF default, `tmf`, for the OpenVault application name or specify a different name for TMF with the following parameter:

   `ov_tmf_application_name =` *tmf_application_name*

   If this line is omitted from the `LOADER` statement, the default application name will be `tmf`.

d. Either use the TMF default instance name or specify a different instance name for TMF with the following parameter:

   `ov_tmf_instance_name =` *tmf_instance_name*

   If you omit this line from the `LOADER` statement, a default instance name of *tmf_node_name.tmf_application_name* will be used, where the value of *tmf_application_name* is taken from the previous substep.

e. If the OpenVault communication security feature is to be enabled, specify the pathname of the key file that will contain the TMF security key (*tmf_key*):

   `ov_tmf_keyfile =` *tmf_keyfile*

   If this line is omitted from the `LOADER` statement, the OpenVault communication security feature will not be used (*tmf_key* equals `none`). For more information on the key file and security keys, see the *OpenVault Operator's and Administrator's Guide*.

2. Make sure the `DEVICE_GROUP` names match the OpenVault drive group names.
3. Make sure the `name` field in each of the `DEVICE` statements match the corresponding OpenVault drive names.

Automatic Volume Recognition

Automatic volume recognition (AVR) is a TMF feature that does the following:

- Allows TMF to recognize volumes mounted on drives prior to them actually being requested by applications

- Allows an operator to direct the mounting of tapes to specific devices

Tape mount messages request that the operator mount a tape on a device in a device group. Upon receiving a message, you locate the tape and choose the device to be used.

The `overcommit` option is an extension to AVR. It allows you to set the number of outstanding mount requests to a number larger than the actual number of tape devices. It gives you additional flexibility in choosing which request to satisfy and on which device.

Note: Only those requests that cannot cause a device to deadlock are allowed into the overcommitted request process.

You may enable or disable the AVR and overcommit options on a global or on a specific device-group basis. Neither option is available to device groups that also contain devices serviced by a tape library (automatic loader).

When a device configured to use AVR is configured up with the `tmconfig`(8) command, a child process called `tmavr` is created to monitor the device and wait for a volume to be mounted. When `tmavr` detects a mounted volume, the label and ring status information is sent to the TMF daemon. If `tmavr` cannot determine the volume label, an operator message is issued for the correct volume information to send to the TMF daemon. The child process waits for the TMF daemon to direct it to exit or look for a new volume to mount.

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 section provides a brief overview of the daemon and interface:

- "Starting and Stopping the Message Daemon" on page 25
- "Message Logs" on page 25
- "Message Daemon Commands" on page 25

Starting and Stopping the Message Daemon

You must have superuser privileges to start or stop the message daemon.

Start the message daemon prior to starting TMF by entering the following command:

```
# /usr/sbin/msgdaemon
```

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

Stop the message daemon by entering the following command:

```
# /usr/sbin/msgdstop
```

For more information, see the `msgdaemon`(8) and `msgdstop`(8) man pages.

Message Logs

All messages are logged by the message daemon as they are received. The logs are kept in the following file:

`/var/spool/msg/msglog.log`

The `/usr/sbin/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 by the `crontab`(1) command.

Message Daemon Commands

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

The `oper`(8) command provides an operator display that 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 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.

The `$HOME/.operrc` configuration file lists the commands to be run as refresh displays and those that require full control of the screen. (`$HOME` is the user's home

directory.) If this file does not exist, the default configuration file (`/etc/tmf/oper`) is used.

Commands not listed in the configuration file are assumed to be nondisplay commands, which are also called *action commands*.

The `msgr`(1) sends action messages to the operator. Action messages that require replies from the operator are primarily tape mount messages, but they may be other types of messages to which users need responses. These messages are logged by the message daemon. An action message is deleted when the operator replies to it or the sender cancels it.

The `msgd`(8) command displays action messages, such as tape mount messages. The `msgrep`(8) command allows the operator to respond to action messages, such as tape mount messages.

Starting and Stopping TMF Automatically

Installing TMF does not enable starting TMF automatically at system startup. To enable automatic startup of TMF and the message daemon, execute the following `chkconfig`(8) command as `root`:

```
# chkconfig tmf on
```

To stop TMF from starting automatically at system startup, execute the following as `root`:

```
# chkconfig tmf off
```

Starting and Stopping TMF Explicitly

To start TMF explicitly, enter the following command:

```
# /usr/sbin/tmdaemon
```

For information about options, see the `tmdaemon`(8) man page.

To stop TMF explicitly, enter the following command:

```
# /usr/sbin/tmstop
```

The `tmstop` command has no options.

TMF Troubleshooting

This chapter discusses the following troubleshooting topics:

Tape Drive or Job Problems

If a tape drive appears to be hung, but the TMF daemon is still responding to commands such as `tmstat`(1) and `tmgstat`(8), you can use the `tmfrls`(8) command to clear the user's tape reservation. If this method does not work, try the `tmclr`(8) command.

If the problem appears to be hardware related, free the user by the preceding method and check the result with the `tmstat`(1) command. Then configure the drive down with the `tmconfig`(8) command and discuss the problem with the appropriate hardware personnel.

TMF Daemon Problems

If the TMF daemon is hung (that is, no tapes are moving nor are there any responses from any tape commands), you must take the TMF daemon down. Do the following, as needed:

1. Use the `tmstop`(8) command.
2. If `tmstop` does not work, determine the process identifier of the TMF daemon (*tmdaemon_pid*) by using the `ps`(1) command and then use the `kill`(1) command:
   a. Interrupt the `tmdaemon` process by entering the following `kill` command:

      # **`kill -2 tmdaemon_pid`**

b. If the previous `kill` command does not work, enter the following to forcefully kill the process:

```
# kill -9 tmdaemon_pid
```

3. Stop TS:

```
# /etc/init.d/ts stop
```

For more information about the TMF daemon, see the `tmdaemon`(8) man page.

Using Tracing

Using tracing can help identify and resolve tape problems. The `tmcollect`(8) utility enables you to collect the trace information needed. This section discusses the following:

TMF Trace Files

During the course of its activity, the TMF daemon and its components write a number of trace files, which are located in the `/usr/spool/tmf/trace` directory. Table 3-1 describes these files.

Table 3-1 TMF Trace Files

File	Description
`avr_`*device_name*	Each `tmavr` process records events in a trace file based on the device name it is monitoring. For example, if AVR is active for the `s4781s0` device, the relevant trace entries for the `tmavr` process are in `avr_s4781s0`.
`daemon`	This file contains all activity traced by the TMF daemon. It is the main TMF daemon trace file.
`daemon.stdout`, `daemon.stderr`	These files contain any information that goes to standard output or error. They are in the `/usr/spool/tmf` directory. The `daemon.stderr` file is especially helpful in tracking down problems because it contains error messages as well as informational messages pertaining to various administrative commands.
ldrname	Each media loader also has its own trace file. The name of this file corresponds to the loader name as defined in the `tmf.config` file.
`tmf`*xxx*	After a tape is assigned a drive, subsequent traces specific to that process are logged in a `tmf`*xxx* file. The final three characters of the trace file can be determined from the `stm` field of the `tmstat`(1) command. "`tmstat` Output" on page 30 shows how you use the `tmstat`(1) command to identify a `tmf`*xxx* file.

`tmstat` Output

In this `tmstat` output, the traces for drive `s4781s0` are in the `tmf002` file. Leading zeros are added to the stream number (`stm`) to make it a 3-character number to create the `tmf`*xxx* file name.

```
% tmstat
               user     sess   group   a stat device   stm rl ivsn   evsn   blks  NQSid
                             STK9490 - idle s9490s4

                             STK9490 - idle s9490s1

               bar      3854   STK4781 - assn s4781s0    2 is 002335 002335     1
                             STK4781 - idle s4781s1

                             STK4781 - idle s4781s2

                             STK4781 - idle s4781s3

                             STK4781 - idle s4781s4
```

In addition, communication pipes are maintained within the `/usr/spool/tmf` directory. If the TMF daemon abnormally terminates, its core file is also saved in the directory.

The message daemon logs can provide insight into tape problems. These log files are generally saved and maintained in the `/usr/spool/msg` directory. All operator interaction is saved in the `msglog.log` file. In addition, a debug log for the message daemon is in the `dbglog.log` file.

`tmcollect` Utility

The `tmcollect`(8) utility collects TMF information. A user with `root` permission may run this script when a tape-related problem occurs. The information is placed in a separate directory so that it can be easily packaged and shipped for offline analysis. For the collected information to be of optimal use, TMF tracing should be enabled. For more information about this administration command, see the `tmcollect`(8) man page.

Before anything is copied to the information directory, the `tmcollect`(8) utility attempts to determine whether the TMF daemon is in its normal state; if it is not, it runs a few checks for known hang situations.

You should use the `tmcollect`(8) utility to gather information if you suspect trouble with the TMF daemon, prior to terminating the TMF daemon.

Enabling and Disabling Tracing

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

```
# tmset -T off
```

To turn tracing on, enter the following command:

```
# tmset -T on
```

If the stability of TMF at a site has been established, tape tracing may be unnecessary overhead. The number of 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 TMF 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 TMF. 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 TMF tracing off allows sites at which TMF is stable to reduce substantially the system and user time used by the TMF 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 TMF daemon problem.

The only way to analyze a problem is to turn tracing on, resubmit the job, and collect traces when the problem reappears.

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 `tmcollect`(8) utility to aid in the data gathering process.

This utility saves all the pertinent trace files in `/var/spool/tmf`. If the TMF daemon is not hung, the TMF command output is also saved. When you execute the utility, you are asked to comment on how the system was behaving at the time `tmcollect`(8) 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 TMF daemon trace has probably been overwritten. However, the device trace should provide some useful information. By default, the device traces are 409600 bytes in length while the `daemon` file is 10 times that value (the default is 4096000 bytes). You can configure this parameter by specifying the `trace_file_size` option in the `OPTIONS` statement in the TMF configuration file. For more information, see the `tmf.config`(5) man page.

Each time a TMF daemon routine is entered, tracing for that routine begins. Additional tracing may also exist that provides more information for software engineering in case problems occur. By using this information, the paths that the software took to perform various tape functions can be followed.

Information is also written into the respective TMF daemon device traces (`tmf`*xxx*). In addition, there are trace files for `esinet`, `stknet`, and `ibmnet`. By using all of the appropriate traces, you can obtain the entire picture of what was happening when a failure occurred.

The following example identifies and describes each trace line segment:

```
10:59:58 151257598.1241 1450 tmmsp media_select function entered
^^^^^^^^ ^^^^^^^^^^^^^^^ ^^^^ ^^^^^ ^^^^^^^^^^^^ ^^^^^^^^^^^^^^^^.......
AAAAAAAA BBBBBBBBBBBBBB CCCC DDDDD EEEEEEEEEEEE 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, 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 `daemon` file, this value will invariably be `tmdaemon`(8); in the `tmf`*xxx* files, the value will be the particular child `tmdaemon`(8) forks off to process the request (for example, `tmmsp`).

D	Identifies the main routine.
E	References the particular routine called by the main routine.
F	Provides detailed trace information about the entry.

Resolving Common Problems

This section identifies some common tape problems that you may encounter and some possible solutions.

TM003 - Resource *group_name* is not available

This error indicates that you issued a tmrsv(1) command for a device group that does not exist or that you attempted to reserve more devices than are currently configured up.

TM060 - Waiting for device *device_name*

This message is returned when a tmmnt(1) command has been issued but has not yet been satisfied because a requested device type is not available. The command will be satisfied after a device is made available either by the operator configuring a device up or by a currently running job releasing its resources.

TM064 - File *file_name* could not be found on volume *vsn*

This error is returned when the file specified with the -f parameter on the tmmnt(1) command (or -p if -f is not present) does not exist on a labeled tape. When a labeled tape is created, the lower 17 characters specified by the -f (or -p) parameter are written into the HDR1 label. Subsequent attempts to read that tape file must include the correct file identifier. The file identifier is not checked if the check_file_id option is set to NO in the tmf.config file.

TMF Commands

This appendix discusses the following:

User Commands

Table A-1 summarizes the TMF user commands in `/usr/bin`.

Table A-1 User Commands in `/usr/bin`

Command	Description
`msgr`(1)	Sends action messages to the operator
`tmcatalog`(1)	Catalogs, recatalogs, or deletes a dataset in a front-end catalog
`tmlist`(1)	Lists the contents of one or more tape volumes
`tmmnt`(1)	Requests that the system operator mount a tape
`tmrls`(1)	Releases reserved tape resources
`tmrst`(1)	Displays reserved resource status for current session identifier
`tmrsv`(1)	Reserves tape resources
`tmstat`(1)	Displays current tape status

Administrator Commands

Table A-2 summarizes the TMF administrator commands in `/usr/sbin`.

Table A-2 TMF Administrator Commands in `/usr/sbin`

Command	Description
`msgd`(8)	Displays action messages
`msgdaemon`(8)	Starts the message daemon
`msgdstop`(8)	Stops the message daemon
`msgrep`(8)	Replies to action messages
`newmsglog`(8)	Saves the latest versions of the message log file
`oper`(8)	Invokes the operator display manager
`tmclr`(8)	Clears a tape stream
`tmcollect`(8)	Collects information for TMF problem analysis
`tmconfig`(8)	Configures tape devices up and down
`tmconf`(8)	Verifies TMF configuration file and converts it to binary format
`tmdaemon`(8)	Starts TMF
`tmfrls`(8)	Forcibly releases tape reservation and associated devices
`tmgstat`(8)	Displays user reservation status for all users
`tmlabel`(8)	Labels a tape
`tmmls`(8)	Displays loader status
`tmmql`(8)	Displays TMF mount request queue list
`tmset`(8)	Displays user reservation status for all users
`tmstop`(8)	Stops TMF
`tmunld`(8)	Unloads tape drives

Glossary

`$HOME`
The user's home directory

ACSLS
Automated Cartridge System Library Software

attended mode
A tape device that is serviced by a manual operator

AVR
Automatic volume recognition

CPS
Controlled Path Service software

device group
A collection of devices with equivalent physical characteristics

DMF
Data Migration Facility

mixed environment
An environment with some devices in *attended mode* and other devices in *unattended mode*

OpenVault
A mounting service that can be used as a tape loader by TMF

ovserver_host_name
Name of the host on which the OpenVault server is running

PCL
Physical cartridge label

physical tape
Physical cartridge

StorageTek
Library using the ACSLS software interface

TMF
Tape Management Facility mounting service

tmf_application_name
OpenVault application name to be used by TMF

tmf_host_name
Name of the host on which TMF will run

tmf_instance_name
OpenVault instance name to be used by this instance of TMF

tmf_key
TMF security key to be used for TMF

tmf_keyfile
OpenVault keyfile to be used by TMF when communicating with the OpenVault server

tmf_node_name
Node name of the host on which TMF will run

VSN
Volume serial number

unattended mode
A tape device that is serviced by a library

Index